Lifetime Reliability of Multi-core Systems: Modeling and Applications

HUANG, Lin

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Master of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

April 2011

Thesis/Assessment Committee

Professor YOUNG Fung Yu (Committee Chair)
Professor XU Qiang (Thesis Supervisor)
Professor LEE Kin Hong (Committee Member)
Professor XIE Yuan (External Examiner)

Abstract

Advancements in technology have enabled the integration of a great amount of processor cores on a single silicon die, known as multi-core systems. Quite a few commercial designs have been launched into market in recent years, for example, nVidia 128-core GeForce 8800 GPU [77] and ARM11 PrimeXsys platform [7]. Some research groups even predicted that thousand-core processors would be commercialized in next decade. When we rejoice in the admirable functionality and performance provided by these products, the ever-shrinking feature size is subtly undermining this apparent flourish, posting serious challenges to the silicon integrated circuit (IC) community.

The ever-increasing on-chip power and temperature density has significantly accelerated the aging effect induced by permanent failure mechanisms of state-of-the-art IC products and hence dramatically shortened products' service life. Thus, it becomes a challenging task for designers to meet multi-core chips' lifetime reliability requirement. This is not a trivial task: because the aging rate depends on operational temperature, supply voltage, and operational frequency, almost all design-stage decisions and even the system workload affect the reliability stress on processor cores. Needless to say, efficient stress estimation is urgently required by designers. Unfortunately, there is neither natural way nor well-accepted literature on this issue.

To tackle this problem, we develop a comprehensive mathematical model that

enables us to achieve general closed-form expression of multi-core system service life. With this model, we then build an efficient yet accurate simulation framework, named *Agesim*, for evaluating the lifetime reliability of processor-based system-on-chips (SoCs). Different from previous works, *AgeSim* traces the reliability-related usage strategy over much shorter time duration without sacrificing the accuracy much. In addition, the wear-out effect induced by failure mechanisms can be modeled with arbitrary failure distributions in *AgeSim*, avoiding the inaccuracy induced by constant failure rate assumption. Multiple case studies are conducted to demonstrate the flexibility and effectiveness of the proposed methodology. Sequentially, the ideas behind the proposed model and simulation framework are applied onto the task allocation and scheduling problem for multi-core embedded systems. Three techniques are finally developed to enhance the system lifetime reliability.

摘要

技術進步使單一矽晶片上可以集成大量處理核心,稱爲多核心系統。近年來不少商業設計被投放市場,例如 nVidia 的 128 核心 GeForce 8800 GPU [76]和 ARM11 PrimeXsys 平臺[7]。一些研究團體甚至預言千核心處理器將在下一個十年商業化。當我們爲這些產品所提供的令人讚歎的功能和性能而欣喜時,不斷縮小的特徵尺寸正在悄悄的破壞這表面上的繁榮,給矽積體電路(IC)領域帶來嚴峻挑戰。

不斷升高的晶片功率密度和溫度密度已大大加速了現今 IC 產品的永久性破壞機理所導致的老化作用,從而大幅縮短了這些產品的使用壽命。那麼,設計能夠滿足壽命可靠性要求的多核心晶片成爲設計者具有挑戰性的工作。這並非平凡的工作:由於老化速率取決於操作溫度、供應電壓和工作頻率,幾乎所有設計階段的決定甚至於系統工作負荷都有可能影響處理核心的可靠性壓力。不用說,有效的壓力估計是設計者所急切需要的。不幸的是,在這一問題上既沒有自然的方法也沒有廣泛接受的文獻。

爲了解決這一問題,我們建立了綜合數學模型來獲得多核心系統使用壽命的一般閉合表達形式。使用這個模型,我們接下來建立了高效而準確的模擬框架,名爲 AgeSim,用以評估基於處理器的片上系統(SoC)的生命週期可靠性。與以往的工作不同,AgeSim 只在較短的時間段內追蹤與可靠性相關的使用策略,而又不大量犧牲精確度。另外,在 AgeSim 中由失效機制導致的磨損作用可以被建模成任意故障分佈,避免了指數分佈假設帶來的不精確性。我們對多個案例進行研究來展示所提出的方法的靈活性和有效性。接下來,所提出的模型和模擬框架背後的思想被應用到多核嵌入式系統的任務分配和調度問題上。我們建立了三種用以提高系統生命週期可靠性的技術。

Acknowledgement

It is my fortunate to meet Professor Xu Qiang and to have him as my supervisor. In the past three years, he continually conveyed a spirit of enterprise and excitement in regard to research, guided me through the difficulties in research, supported me with his research funding, and taught me how to write up research papers and deliver presentations. All are essential to the final completion of this thesis, and I would like to express the deepest appreciation to him. In personal feeling, I do love the moments of sharing my happiness and achievement with him and those of inspiration from our discussions. Without a doubt, the completion of this thesis would not have been possible without his great effort.

I gratefully thank my markers, Professor Evangeline F.Y. Young and Professor Lee Kin-Hong, whose constructive comments in term presentations gave such an important impetus to my research progress. In particular, I am so grateful for the bright sunshine brought by Professor Evangeline F.Y. Young, who is always nice and willing to help.

I, also, would like to thank Prof. Xie Yuan for serving as external marker and reading my thesis.

It is a pleasure to pay tribute also to my collaborators. To Yuan Feng, I would like to thank him for his insightful ideas that promoted our research progress and experiment work that proved the effectiveness of our tentative ideas. To Zhang Yubin, I would like to thank him for being the first person who helped me to orient

myself in CUHK and the person who looked for an apartment for me before my arrival in Hong Kong. To Tang Matthew, thank you for teaching me how to use the online resources of CSE department. To Ye Rong and Jiang Li, it is a pleasure to collaborate with you and I wish you all the best in your future career. To Shi Lei, I am thankful for always helping me out of mathematics problems. I would also acknowledge Yu Haile, Zhang Ji, and Liu Yuxi for their advice and their willingness to share their bright thoughts with me. Thank you, my dear friends in Rm 506, Ho Sin Hang Engineering Building: Xiao Linfu, Qin Jing, Ma Qiang, Yang Xiaoqing, Jiang Yan, Jiang Mingqi, Qian Zaichen, Li Liang, and Tam Tak Kei. My life in CUHK becomes abundant and colorful because of you.

I would also acknowledge Tsang Kin Lun (Calvin) and Pih Wing Yin (Annie) for their great effort in the maintenance of an efficient working atmosphere and the Christmas parties.

I was very fortunate in having Professor Yuan Yan as my advisor at Shanghai Jiao Tong University. I could never have embarked on my journey towards academic study without his prior guidance.

My special thanks to my parents, who sincerely raise and support me with their precious love. Whatever happens, they always bring me the constant patience, endless caring, and gentle encouragement. My grandmother deserves my heartfelt thanks for her everlasting love. I am deeply sorrow that I have no opportunity to share with her the joy of graduation. This thesis is completed in memory of her. Last but not least, I would like to express my grateful appreciation to my boyfriend Liu Xiao for his genuinely love, glowing passion, and persistent confidence in me. In the life journey one need to carry so many burdens. In the past years, he has taken off a great share of mine from my shoulder and support me silently.

I dedicate this thesis to the memory of my grandmother.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preface

This thesis includes, but is not limited to, my research work in the past three years that have been published in conference proceedings and journals. Because of the space limitation, this thesis does not cover the basic concepts in probability theory and basic failure models. Interested readers can refer to [106] for a more mathematical text and [34] for an introduction to reliability engineering.

The thesis consists of three parts. The first part covers the lifetime reliability modeling of multi-core systems. Chapter 2 builds the mathematical foundation of entire thesis. I therefore highly recommend covering this chapter – through Section 2.5, at least – to achieve better understanding of multi-core systems.

The second part is concerned with the simulation framework that is used to evaluate the lifetime reliability of various systems. Chapter 3 targets a key issue in reliability engineering, that is, the gap between theoretical modeling and real world applications. The proposed framework *AgeSim* enables an efficient yet accurate evaluation of single- or multi-core systems with any dynamic power management (DPM) or dynamic thermal management (DTM) policies, application flow char-

acteristics, and even task allocation algorithms. We then develop an application of *AgeSim* in Chapter 4 and show some interesting observations in experimental results, emphasizing the impact of redundancy schemes on system lifetime reliability. I recommend reading Chapter 3 because of its importance for the subject and then selecting sections from Chapter 4 according to time and interests.

The third part brings the mathematical model and simulation framework into an real world application: task allocation and scheduling for multiprocessor system-on-chips (MPSoCs). Chapter 5, 6, and 7 explicitly bring the lifetime reliability of MPSoCs into the task allocation and scheduling process, each targeting a specific optimization objective under a specific set of constraints. The proposed solutions varies according to the characteristics of problems. Among these three chapters, Chapter 5 is the first comprehensive work on this topic and hence deserves the highest reading priority.

Every chapter in this thesis is largely self-contained. One can read any chapter without going through previous chapters beforehand. Yet I need to note that the notations defined in each chapter is applicable for that chapter only.

Lifetime reliability modeling part includes the following publications:

- **Lin Huang** and Qiang Xu, "Lifetime Reliability for Load-Sharing Redundant Systems with Arbitrary Failure Distributions", IEEE Transactions on Reliability, vol. 59, no. 2, pp. 319 - 330, Jun. 2010.

- **Lin Huang** and Qiang Xu, "On Modeling the Lifetime Reliability of Homogeneous Manycore Systems", Proc. Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 87 - 94, Dec. 2008.

Simulation framework part includes:

- **Lin Huang** and Qiang Xu, "AgeSim: A Simulation Framework for Evaluating the Lifetime Reliability of Processor-Based SoCs", Proc. IEEE/ACM

Design, Automation, and Test in Europe (DATE), pp. 51 - 56, Mar. 2010. **(Best Paper Nomination)**

- **Lin Huang** and Qiang Xu, "Characterizing the Lifetime Reliability of Many-core Processors with Core-Level Redundancy", accepted for publication in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2010. **(Best Paper Nomination)**

Applications of lifetime reliability enhancement part includes:

- **Lin Huang**, Rong Ye and Qiang Xu, "Customer-Aware Task Allocation and Scheduling for Multi-Mode MPSoCs", accepted for publication in Proc. ACM/IEEE Design Automation Conference (DAC), Jun. 2011.

- **Lin Huang** and Qiang Xu, "Energy-Efficient Task Allocation and Scheduling for Multi-Mode MPSoCs under Lifetime Reliability Constraint", accepted for publication in Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), pp. 1584 - 1589, Mar. 2010.

- **Lin Huang**, Feng Yuan and Qiang Xu, "Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms", Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), pp. 51 - 56, Apr. 2009.

Apart from the above articles, there are many publications that will not be touched in this thesis, including:

- **Lin Huang** and Qiang Xu, "Yield Analysis and Redundancy Allocation for Multi-core Chips", not submitted yet.

- **Lin Huang** and Qiang Xu, "Asymmetry-Aware Processor Allocation for NoC-Based Chip Multiprocessors", not submitted yet.

- **Lin Huang** and Qiang Xu, "Economic Analysis of Testing Homogeneous Many-core Chips", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29, no. 8, pp. 1257 - 1270, Aug. 2010.

- **Lin Huang** and Qiang Xu, "Performance Yield-Driven Task Allocation and Scheduling for MPSoCs under Process Variation", Proc. ACM/IEEE Design Automation Conference (DAC), pp. 326 - 331, Jun. 2010.

- **Lin Huang**, Feng Yuan and Qiang Xu, "On Reliable Modular Testing with Vulnerable Test Access Mechanisms", Proc. ACM/IEEE Design Automation Conference (DAC), pp. 834 - 839, Jun. 2008.

- **Lin Huang** and Qiang Xu, "Test Economics for Homogeneous Many-core Systems", Proc. International Test Conference (ITC), Paper 12.3, Nov. 2009.

- **Lin Huang** and Qiang Xu, "Is It Cost-Effective to Achieve Very High Fault Coverage for Testing Homogeneous SoCs with Core-Level Redundancy?", Proc. International Test Conference (ITC), Oct. 2008.

- Li Jiang, **Lin Huang** and Qiang Xu, "Test Architecture Design and Optimization for Three-Dimensional SoCs" Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), pp. 220 - 225, Apr. 2009.

- Feng Yuan, **Lin Huang** and Qiang Xu, "Re-Examining the Use of Network-on-Chip as Test Access Mechanism", Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), pp. 808 - 811, Mar. 2008.

- Yubin Zhang, **Lin Huang**, Feng Yuan and Qiang Xu, "Test Pattern Selection for Potentially Harmful Open Defects in Power Distribution Networks", Proc. IEEE Asian Test Symposium (ATS), pp. 460 - 465, Nov. 2009.

1.2 Background

You may have noticed that we are entering the multi-core era, which is characterized by the integration of a great amount of processor cores on a single silicon die. There is an ever-increasing influx of multi-core designs that are built in recent years into our vision and commercial market. Examples include Tilera 64-core TILE64 processor [11], nVidia 128-core GeForce 8800 GPU [77], Cisco 192-core Metro network processor [25], and Intel 80-core teraflop processor [108]. Most of these designs employ core-level redundancy (in addition to module-level redundancy) to enhance the fault-tolerant capability, because a single processor core is not so expensive as before when compared with entire chip. Cisco Metro network processor [25], for instance, contains four redundant cores, which are identical with the remaining 188 active cores.

While advancements in semiconductor technology have brought with unprecedentedly high functionality and performance, the appendant challenges, especially the ever-increasing adverse impacts on the reliability of IC products, have become nightmare of design engineers. One of the most daunting challenges is the ever-shrinking service life. It has been observed that the lifetime of IC products shortens from 10 years to 7 years, to less than 7 years as the feature size reduces from $180nm$ to $130nm$, to $90nm$. This phenomenon can be attributed to the fact that the increasing on-chip power and temperature density accelerate the wear-out effect induced by multiple failure mechanisms. The most representative failure mechanisms are electromigration (EM) on the interconnects, time-dependent dielectric breakdown (TDDB) in the gate oxides, thermal cycling (TC), and negative bias temperature instability (NBTI) on PMOS transistors. Most of them are not recoverable and therefore could result in permanent errors once they manifest. NBTI, which manifests as an increase in the threshold voltage, is an exception. PMOS transistors enter the recovery phase under certain condition, but this process is pretty slow

and hence not likely to achieve 100% recovery.

This phenomenon has attracted a great deal of attention from both academics and industry. Unfortunately, the development of comprehensive understanding is not significant in the past years. This is primarily caused by the difficulties of developing comprehensive mathematical models that should be able to capture all main features of multi-core computing systems, and the difficulties of bridging the analytical models towards real world applications.

This thesis exhaustively explores this problem. In the rest of this chapter, a brief overview of the main contributions is presented.

1.3 Contributions

1.3.1 Lifetime Reliability Modeling

Whereas that things fail is quite common in our daily life, hardly any persons, including experts, can tell the probability of an object failing two hours later. This requests in-depth insights into failure mechanisms and realistic models of the object. Both are indispensable. Thanks to the researches in device and material, who have revealed the fundamentals of most failure mechanisms in semiconductor devices. Yet, until a few years ago, lifetime reliability remains an impenetrable issue in the state-of-the-art multi-core system design due to the lack of realistic, high-level models.

Previous work on modeling is basis on a series of questionable assumptions. In particular, most assume the failure distribution that is used to describe the time dependence of the failure rate as exponential distribution. This assumption implies that IC products experience constant failure rate through their lifetime, which is irreconcilable with the actual situation, that is, ICs suffer from wear-out effect induced by multiple failure mechanisms. The prime reason for the popularity of this

assumption is its mathematical tractability rather than accuracy. To take another example, some assume the cores that are configured as active ones are always busy and hence follow a certain failure distribution while the remaining cores (i.e., redundant cores) never fails because they do not share the system workload. The model based on this assumption cannot explain the phenomenon that heavy system workload results in high reliability stress on hardware. Again, this assumption simplifies the complexity of problem but sacrifices the model accuracy considerably. In reality, each processor core can be in more than one state, depending on its current workload. Each state, in addition, corresponds to its own non-exponential failure distribution. From the system point of view, the amount of processor cores in a certain state depends on the system workload and load-sharing strategy. Besides, Since the processor cores share the system workload, the redundancy scheme and permanent core failures in the system affect the current workload of surviving cores.

We now develop a mathematical model that abstracts multi-core systems with core-level redundancy as load-sharing k-out-of-n:G hybrid redundant system and achieves the closed-form expression of lifetime reliability, capturing all key features of multi-core systems. Moreover, we resort to Monte Carlo simulation to approximate the multiple integrals in the closed-form expression. This model allows us to comprehensively understand the aging process of multi-core systems and builds the mathematical foundation of entire thesis. In addition, we numerically demonstrate the misleading results achieved by assuming exponential failure distribution.

1.3.2 Simulation Framework

The mathematical model mentioned above is one step away from the multi-core designs, yet this step is undoubtedly necessary for designers to meet the reliability

requirement. To clarify, we bring the assumed failure distribution for each state into the model. To evaluate the impact of design-stage decisions, these failure distributions should be able to reflect the influence of reliability-related policies on the system lifetime. Due to the lack of nature bridge, without an efficient yet accurate simulation framework, it is extremely difficult, if not impossible, to make the right decisions. However, since the reliability stress on processors vary significantly at runtime, it is a challenging task to build such a simulation framework. Obviously, it is not acceptable to trace all the reliability-related factors over lifetime (in the range of years) and use the traced data for simulation. Nor is it acceptable to approximate the lifetime reliability with average operational temperature. The former one is too time-consuming, while the latter one is lack of accuracy. The problem becomes even challenging when the wear-out effect that results in increasing failure rate is taken into account.

To overcome these difficulties and facilitate the design process, for the first time we propose to wrap the impact of reliability-related usage strategies into a single quantity named *aging rate*, and build the simulation framework based on this novel concept. Moreover, for the first time the efficient yet accurate evaluation is achieved by simulating representative workload. The accuracy of this framework is theoretically proved in this thesis. The proposed framework enables us to evaluate various dynamic power management policies, timeout policies, application flow characteristics, redundant schemes, and even task allocation algorithms in single- or multi-core systems. In addition, the proposed simulation framework can output the performance and energy consumption without extra effort, facilitating designers to evaluate systems in various aspects.

1.3.3 Applications

Task allocation and scheduling process, which results in specific processor cores being engaged in specific tasks in specific order, is an important step in deploying applications on MPSoCs. Different task schedules can result in significantly different reliability stress on processor cores. We then bring the mathematical model and simulation framework into this real world application. While there are a few publications on reliability-aware task allocation and scheduling, most attempt to minimize the peak operational temperature or balance different processor cores' temperature and none explicitly takes the lifetime reliability into account. Because lifetime reliability of a system depends on not only operational temperature but also other factors (e.g., supply voltage and operational frequency), these techniques might be implicitly helpful for lifetime reliability enhancement, but they cannot thoroughly solve the problem.

We develop the first task allocation and scheduling algorithm that explicitly take the wear-out effect induced by failure mechanisms into account, within which the optimization objective is set as maximizing the expected service life. In this work, the failure distribution is assumed to be the widely-accepted Weibull distribution rather than exponential distribution. Four speedup techniques are also introduced to achieve efficient lifetime reliability estimation. Experiments on various task graphs and various platforms demonstrate the efficiency of the proposed algorithm and speedup techniques.

Later, the problem is reconsidered in the context of multi-mode energy-efficient embedded systems, where a set of feasible task schedules are constructed for each mode beforehand and then the best combination of schedules is selected and applied in the design.

With the realization of usage strategy deviation of multi-mode embedded systems, we then develop a "smart" technique to online adapt task schedules for dif-

ferent users. The allocation and scheduling process at design stage, albeit carefully conducted, results in optimized schedules with respect to a hypothetical common case at best rather than someone's personalized usage strategy. With this observation, we propose to generate an initial task schedule for each mode and then conduct online adjustment for each particular survival chip at regular interval based on its own usage strategy.

1.4 Thesis Outline

Part I, "Modeling," develops mathematical model useful in analyzing the lifetime reliability of homogeneous multi-core systems with redundancies. Chapter 2 discusses the component behavior and multi-core system behavior, and develops a mathematical model to capture their features. This model is applicable for arbitrary failure distributions.

Part II, "Simulation Framework," is concerned with bridging design-stage decisions and mathematical model. Chapter 3 develops a efficient yet accurate simulation framework to evaluate various reliability-related usage strategies in multi-core systems. The applicability of the proposed simulation framework on evaluating various redundancy schemes is carefully examined in Chapter 4.

Part III, "Applications," presents the development of lifetime reliability-aware task allocation and scheduling for MPSoCs. Chapter 5 develops a static task allocation and scheduling strategy that explicitly takes the aging effect into account. Four speedup techniques are also presented to achieve an efficient lifetime estimation with satisfactory solution quality. Chapter 6 presents a task schedule and scheduling technique for energy-efficient multi-mode embedded systems. This technique is composed of two steps: a set of schedules are first identified for each mode, then the optimal combination is constructed. Different from previous two algorithms, the technique discussed in Chapter 7 is not for design stage only. At

design stage, an initial schedule is generated for each mode. Sequentially, online adjustment is conducted for each particular survival chip based on its past usage strategy. Chapter 8 finally summarizes this thesis and points out the next steps.

□ End of chapter.

Part I

Modeling

Chapter 2

Lifetime Reliability Modeling

The content of this chapter is included in *IEEE Transactions on Reliability* 2010 [50] and the proceedings of *Pacific Rim International Symposium on Dependable Computing (PRDC)* 2008 [46].

2.1 Notation

n	number of components in the system
m	number of active components in the standby phase
k	minimum number of components required for system operation
$\|S\|$	number of elements in set S
λ	task arrival rate of the entire system
μ	task service rate of an active component
ρ	utilization ratio of an active component

p	an active component's task branch-out probability
$\mathcal{R}(t)$	general reliability function
$R_p(t)$	reliability function of processing state
$R_w(t)$	reliability function of wait state
θ	general scale parameter
θ_p	scale parameter of processing state
θ_w	scale parameter of wait state
t^b	birth time of a component
$\mathbf{t}_\ell$	vector representing occurrence time of past ℓ component failures, $(t_1, t_2, \cdots, t_\ell)$
$\mathbf{t}_\ell^{(r)}$	r-dimensional subvector of $\mathbf{t}_\ell$, $(t_1, t_2, \cdots, t_r)$
$\psi_p(t, t^b, \mathbf{t}_\ell)$	cumulative time of a component with birth time t^b in the processing state having failures at $\mathbf{t}_\ell$
$\psi_w(t, t^b, \mathbf{t}_\ell)$	cumulative time of a component with birth time t^b in the wait state having failures at $\mathbf{t}_\ell$
$\psi(t, t^b, \mathbf{t}_\ell)$	unified cumulative usage time of a component with birth time t^b having failures at $\mathbf{t}_\ell$
$P^{sys}(t)$	system reliability at time t
$P_{j,i}^{sys}(t)$	the probability that the system contains j active components, and i good spare components at time t
$P_{n-\ell}^{sys}(t)$	the probability that the system contains $(n-\ell)$ good components at time t

$P_{n-\ell}^{sys}(t, \mathbf{t}_\ell; \mathbf{x}_\ell)$	the probability that the system contains $(n-\ell)$ good components at time t, and ℓ failures can be described by vectors $\mathbf{t}_\ell$, and $\mathbf{x}_\ell$
$P_{n-\ell}^{sys}(t \mid \mathbf{t}_\ell; \mathbf{x}_\ell)$	the conditional probability that the system contains $(n-\ell)$ good components at time t given the ℓ failures can be described by vectors $\mathbf{t}_\ell$, and $\mathbf{x}_\ell$
$R(t, t^b \mid \mathbf{t}_\ell)$	conditional reliability, that is the probability that a component with birth time t^b survives at time t given the system experiences ℓ failures at $\mathbf{t}_\ell$ respectively $(t > t_\ell)$
$f(t_r, t_{x_r} \mid \mathbf{t}_\ell^{(r-1)})$	the probability that an indicated component with birth time t_{x_r} fails at time t_r given the past $(r-1)$ failures of the system occurs at $\mathbf{t}_\ell^{(r-1)}$
$\mathbf{x}_\ell$	ℓ-dimensional vector representing the birth time indices of past ℓ component failures, $(x_1, x_2, \cdots, x_\ell)$
$\mathbf{x}_\ell^{(r)}$	r-dimensional subvector of $\mathbf{x}_\ell$, $(x_1, x_2, \cdots, x_r)$
$\mathcal{X}_\ell$	set of all possible $\mathbf{x}_\ell$
$\pi_{i,j}$	the number of i in the first j elements of vector $\mathbf{x}_\ell$
$g_{n-r+1}^{sys}(t_r; x_r \mid \mathbf{t}_\ell^{(r-1)}; \mathbf{x}_\ell^{(r-1)})$	the probability that a system containing $(n-r+1)$ good components experiences the r^{th} failure at time t_r, and the failure component has birth time t_{x_r} given the past $(r-1)$ failures can be described by $\mathbf{t}_\ell^{(r-1)}$ and $\mathbf{x}_\ell^{(r-1)}$

$MTTF^{sys}$ system mean time to failure

2.2 Assumption

1. The system is a hybrid k-out-of-n:G system.
2. All components are s-independent.
3. A component is in either active mode, or spare mode (as cold standby). An active component alternates between the wait state (as warm standby), and the processing state (as operating component).
4. All active components in this system share the load equally.
5. The failure-time distributions for both warm standby, and operating components follow an arbitrary baseline reliability function. They differ in terms of their scale parameter. Components in cold standby have a zero failure rate.
6. No repair or maintenance is considered.
7. Switching components is a perfect process.

2.3 Introduction

In many load-sharing systems, the load to be processed is specified as tasks, such as applications performed by processing elements in a multiprocessor computing system, bar codes read by laser scanners, or cars assembled by robots. The time scale for processing a single task is usually much smaller than that of a system's lifetime. Therefore, depending on whether or not a component is performing tasks, it may frequently alternate between the *processing state*, and the *wait state* in its

lifetime, as depicted in Fig. 2.1. Generally speaking, components operate at higher temperature, higher pressure, and/or higher speed, and hence will wear out more quickly in the processing state than in the wait state. Consequently, it is more reasonable to regard components in the wait state as in warm standby, when compared to prior work that essentially assumes hot standby [72, 86]. In addition, a system may contain some *spare components* to provide fault tolerance, which converts into *active mode* (including processing and wait states) when an active one fails. If no more spare components exist in the system, when an active component fails, the system will assign more tasks on the surviving components in unit time, which can increase their failure rate.

Without loss of generality, we consider hybrid redundant k-out-of-n:G systems [75, 106], in which m are initially set as active units, with the remaining $(n-m)$ components put aside ($k \leq m \leq n$). Upon detection of the failure of an active component, the system attempts to replace the faulty one with a spare one until there is no spare component in the system. This process is called the *standby*

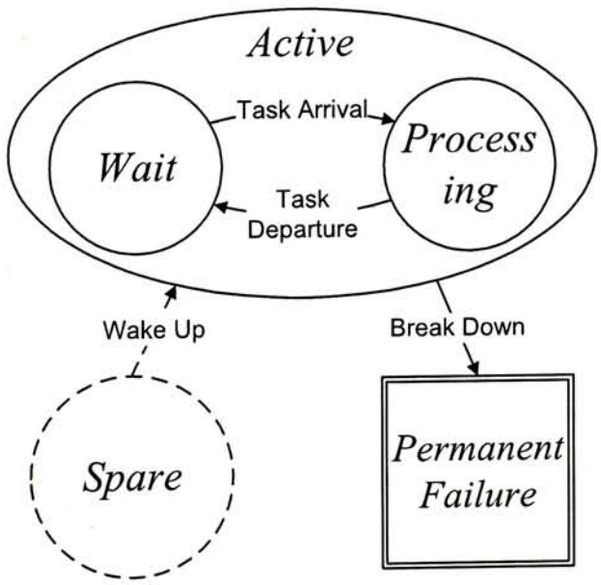


Figure 2.1: The Component Behavior of Hybrid Redundant Systems [46].

phase. We assume that a dedicated switching component takes charge of system reconfiguration and this process is perfect. Because then the system works in a gracefully degrading manner, we refer to this phase as the *degrading phase.* That is, when a component failure is detected, the system attempts to reconfigure to a system with one fewer component but k active components, until no more than k good components are left in the system, all of which are active. Such a system functions correctly if at least k out of the n components do not fail. When $n = m$, the system discussed above becomes a load-sharing k-out-of-n:G gracefully degrading system; while when $k = m$, it is essentially a standby redundant system.

It is rather challenging to model the lifetime reliability of the above system. First, an active component can be in two states: processing and wait. While the quantity of active components in the system is clear, that of components in each state at a particular moment depends on the current workload, and therefore is uncertain. Second, each state corresponds to its own failure distribution. To express the reliability of a component, and the entire system, we need an integrated failure distribution. Note that, in most cases, we cannot predict the exact arrival time, and service time of tasks (the only exception is deterministic arrival, and deterministic service). Therefore we do not know when a component transitions between processing and wait states. Moreover, the frequency of state transitions can be quite high, which also brings challenges to achieve an integrated failure distribution. This problem becomes even more complicated when these failure distributions are not exponential. Last but not least, because active components need to share the load, failures may result in higher workload on the surviving ones, and hence affect their failure distributions. The model should also be able to capture this fact.

To tackle the above problem, in this paper, we develop an analytical model that captures the complex relationship discussed above. We introduce the *cumulative time* concept to reflect the aging effect in each state with arbitrary failure distri-

butions. Next, the cumulative time in all states are combined in a unified manner to express the reliability function of a single component. After that, we model the lifetime reliability of the entire system that involves the task allocation mechanism and redundancy strategies. Then, we discuss several special cases in detail to show the practical applicability of the proposed technique.

The reminder of this paper is organized as follows. In Section 2.4, we survey related work. Section 2.5 then details the proposed reliability model. Next, we verify the proposed models, and demonstrate the detailed analytical procedure with three special cases in Section 2.6. We then present a series of numerical experimental results obtained with the proposed modeling method, and Monte Carlo integration in Section 2.7. Finally, Section 2.8 concludes this paper.

2.4 Related Work

As highlighted in [72], the key feature of load sharing k-out-of-n:G systems is that workload has significant influence on every component's failure rate. While there are many studies on this topic capturing this feature, most of them assume an exponential lifetime distribution for every component [72, 86, 70]. By this assumption, the entire system can be represented by a Markov transition diagram, and hence the complexity analysis comes down to a relatively simple problem by applying mature techniques. For example, assuming all functioning components' failure rates are the same constant at any time, and depend on the number of functioning components in the system, [86] models a load-sharing k-out-of-n:G system by a discrete-state, continuous-time homogeneous Markov chain, and solves its differential equations by inverse Laplace transform.

This assumption may be applicable for some special cases, such as modeling soft errors in IC products, but is obviously not always true. Consider a brand-new unit, and a 10-year old one. In reality, we usually expect their failure rates to be dif-

ferent. The exponential failure distribution assumption, however, implies that the failure rate stays the same after 10 years usage. The main reason for the popularity of the above assumption is its mathematical tractability rather than accuracy. To tackle this problem, [42] studied a 1-out-of-2:*G* system with time-varying failure rates, whose failure distribution can be expressed in a general polynomial format. For a deeper understanding, [72] proposed an analytical model for load-sharing *k*-out-of-*n*:G system with a general lifetime distribution. This work modeled the load on a component as a vector $\mathbf{z}$, and the effect of load as $\psi(\mathbf{z})$. They simply assumed that the effect of load is multiplicative in time without any justification. Also, in this work, the load is assumed to be z_1 initially, and it progressively changes to z_i, after the $i-1$ failure occurs. Each load corresponds to a unique failure distribution. Thus, [72] does not involve warm standby, and the corresponding state transitions. Moreover, although this paper claimed that the proposed approaches can be easily generalized to $n > 2$ systems, its main contribution is limited to load-sharing 1-out-of-2:G systems, and gracefully degrading *k*-out-of-*n*:G systems with equal shared loads. To reduce the computational complexity induced by multiple integrals for such a system, [5] proposed a novel method that transfers the complex calculation to two-dimensional integrals. This work is quite efficient for analyzing load-sharing gracefully degrading systems, but it is difficult to be applied to analyze standby redundant systems. More related works were summarized in [6, 67, 110].

Another issue relevant to this work is how to model the idle components. Such a component can be regarded as a cold, hot, or warm standby unit, which has a zero failure rate, the same failure rate as active components, or a failure rate in between, respectively. Because of simplicity, hot, and cold standby are commonly assumed states in many related papers, which are summarized in [89]. Also, the models mentioned above (e.g., [72, 42]) assumed every component in the system

conforms to a single failure distribution, and hence can only be applied to analyze systems with hot standby components. As discussed earlier, warm standby is clearly a more reasonable description for the reliability analysis in many cases, and hence it is chosen in this paper. [24] provided an in-depth discussion of warm standby. Later, most of the work in this area considered two-unit warm standby systems. For instance, [102] analyzed a two-unit standby redundant system in which a module can alternate between cold and warm standby states; [107] analyzed the two-unit standby system with general lifetime distributions. Similar to many previous work on this topic, it is difficult to extend these models to be applicable for general k-out-of-n systems because of the calculation complexity. As for k-out-of-n warm-standby systems, [88] provided a closed-form expression for the k-out-of-n:G systems with warm standby components, but its analysis is again based on the assumption of constant failure rates in both active, and standby states.

In [40], the authors examined a 1-out-of-3 system that includes a warm, and a cold standby unit. Recently, another mixture model is presented in [117]. This work aimed to handle k-out-of-$(M+N)$:G repairable warm standby systems that consist of two different types of components, each having its own state sets: M type 1 units, and N type 2 units. The operative failure rates, and standby failure rates of type 1, and 2 are different yet assumed to be exponential.

2.5 System Model

In this section, we build our analytical model of lifetime reliability for hybrid redundant load-sharing system. We first examine the behavior of a single component in such a system, and construct the unified reliability function accordingly. And then, we investigate the lifetime reliability of the entire system. Note that, because of frequent mode transitions between wait and processing states, the quantity of components in each state at a particular time point depends on not only previous

failure events but, also current workload. In this context, to capture the variation of system reliability with time becomes a nontrivial problem.

2.5.1 Reliability of A Surviving Component

Component Behavior

Consider a component in the hybrid k-out-of-n:G system. Its initial state can be *active*, or *spare*, as shown in Fig. 2.1. The *spare* mode corresponds to the lowest power consumption, and no interaction with other components or controller. To be specific, a component in the spare mode does not undertake any task. The *active* mode includes two states depending on whether a component has tasks to perform: *processing*, and *wait*. We assume tasks are assigned by a controller, and then performed by a component independently; we ignore the cases that a few components cooperate on a task. If a task is assigned to a busy component (which means this component is processing tasks), it will be stored in a first-in-first-out (FIFO) buffer with infinite capacity. Once a component finishes its task at hand, it will fetch a new one from the queue immediately, unless the queue is empty. In that case, this component will switch from processing to a wait state. Upon receiving a new task, a waiting component will enter a processing state again. Note that, although a component does not process any task in the wait mode, different from the spare mode, it grows older. In reality, consider Intel's StrongARM SA-1100 processor [57] as an example. Its power consumption in a processing state is $400mW$, and that in the wait state is $50mW$ rather than $\sim 0mW$, because some parts are still powered. When an active component fails, if there are some spare components in the system, one of them will be configured to active mode. From then on, it will serve as an active unit, and share workload with other parts until it fails, or the entire system breaks down.

Load-Sharing Model

As mentioned before, workload has significant influence on a component's reliability. Thus, it is necessary to model for each component the load ρ to be used in the reliability function construction. Remind that the tasks are assigned to all active components with equal probability. Thus, given the set of active components in the system $\mathcal{S}_1$, an active component's task branch-out probability is given by $p = \frac{1}{|\mathcal{S}_1|}$. For the sake of completeness, we also define $\mathcal{S}_2$ as the set of spare components, and $\mathcal{S}_3$ as the set of faulty components. The union of these three sets $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 = \mathcal{S}$ forms the entire system, where $|\mathcal{S}| = n$. Although our method could be easily extended to other queueing models (such as a $M^k/M/|\mathcal{S}_1|$ queue for central task assignment with bulk task arrival discussed in [46]), for ease of discussion, we focus on the distributed task allocation mechanism, and model each component as an $M/M/1$ queueing system (as shown in Fig. 2.2). To clarify, the task arrivals to the system are assumed to be Poisson with rate λ, and each component maintains a queue, where a FIFO buffer with infinite capacity is assumed. Because the probability of a task to be executed by an active component (i.e., p) is $\frac{1}{|\mathcal{S}_1|}$, the task inter-arrival time for an active component is exponentially distributed with mean $\frac{|\mathcal{S}_1|}{\lambda}$. Further, assuming the service time is exponentially distributed with mean $\frac{1}{\mu}$, the probability that an active component is occupied by a task, i.e., utilization ratio, is given by $\rho = \frac{\lambda}{\mu|\mathcal{S}_1|}$. For hybrid k-out-of-n:G systems, the utilization ratio of active components is constant $\frac{\lambda}{m\mu}$ in the standby phase, and then gradually increases to $\frac{\lambda}{k\mu}$ at the end of the degrading phase.

Unified Reliability Function

We now introduce our approach to combine the reliability functions in the processing state with those in the wait state, in a unified manner. The reliability functions in these two states can be regarded as having the same shape, but different scale

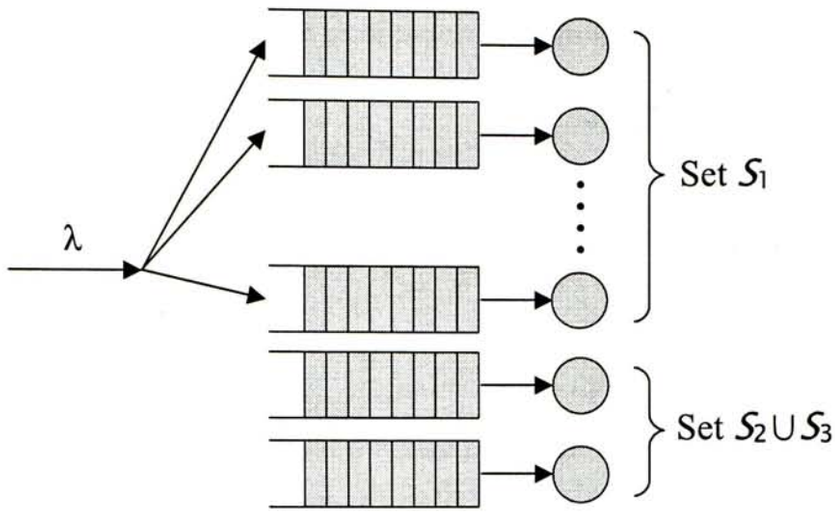


Figure 2.2: Queueing Model for Task Allocation in a Load-Sharing System.

parameters, which is defined as a value by which t is divided, because in many cases they can be distinguished by different aging rates.

First of all, we introduce the concept *cumulative time* in a certain state up to time t, which is defined as how long a component has spent in such a state from time 0 to t. Note that, as we ignore the aging effect in spare state, we are only interested in cumulative time in the processing, and wait states. Recall that some components initially serve as spare units. For the ease of discussion, we define another concept, a component's *birth time* t^b, as the time point when it begins to serve as an active component. Before birth time t^b, a component is in the cold standby mode, and has negligible failure rate. But after that, it alternates between wait and processing states until breaks down. Specifically, the components initially configured as active have birth time $t_0(=0)$.

Theorem 1 *Suppose the system has experienced exactly ℓ failures before time t, in the order of occurrence time at $t_1, t_2, \ldots, t_\ell$ (denoted as vector $\mathbf{t}_\ell$), for any surviving component with birth time t^b.*

(a) Its cumulative time in the processing state up to t ($t > t_\ell$) is

$$\psi_p(t, t^b, \mathbf{t}_\ell) \approx \begin{cases} \frac{\lambda}{m\mu}(t - t^b), & \ell \leq n - m \\ \frac{\lambda}{(n-\ell)\mu}t - \frac{\lambda}{m\mu}t^b & \\ - \sum_{j=n-m+1}^{\ell} \frac{\lambda}{(n-j+1)(n-j)\mu}t_j, & \ell > n - m \end{cases} \tag{2.1}$$

(b) Its cumulative time in the wait state up to t ($t > t_\ell$) is

$$\psi_w(t, t^b, \mathbf{t}_\ell) \approx \begin{cases} (1 - \frac{\lambda}{m\mu}) \cdot (t - t^b), & \ell \leq n - m \\ (1 - \frac{\lambda}{(n-\ell)\mu})t - (1 - \frac{\lambda}{m\mu})t^b & \\ + \sum_{j=n-m+1}^{\ell} \frac{\lambda}{(n-j+1)(n-j)\mu}t_j, & \ell > n - m \end{cases} \tag{2.2}$$

The proof of the above theorem is illustrated in the Appendix.

With the cumulative time in two active states, we perform the integration as follows. The general reliability function which provides us the function shape is defined as $R(t, \theta)$, where θ is the general scale parameter. We drop the notation θ because of its generality, and refer to the general reliability function as $\mathcal{R}(t)$ in the rest of this paper. The functions in processing, and wait states can be therefore expressed as $R(t, \theta_p)$, and $R(t, \theta_w)$ respectively, where the scale parameters θ_p, and θ_w represents the wear-out rate under the two conditions. Typically, $\theta_w \geq \theta_p$. For the sake of simplification, they are abbreviated as $R_p(t)$, and $R_w(t)$. By unification, we achieve two simple relationships between these functions: $R_p(t) = \mathcal{R}(\frac{\theta}{\theta_p} \cdot t)$, and $R_w(t) = \mathcal{R}(\frac{\theta}{\theta_w} \cdot t)$, which enable us to perform reliability function integration for any surviving component with the help of the cumulative time obtained by Theorem 1.

Theorem 2 *Given a system has experienced ℓ failures which occur at $t_1, t_2, \ldots, t_\ell$ (i.e., vector $\mathbf{t}_\ell$) respectively, the probability that a certain component with birth time t^b survives at time t ($t > t_\ell$) can be computed as*

$$R(t, t^b | \mathbf{t}_\ell) = \mathcal{R}\left(\psi(t, t^b, \mathbf{t}_\ell)\right) \tag{2.3}$$

where,

$$\psi(t, t^b, \mathbf{t}_\ell) = \frac{\theta}{\theta_p}\psi_p(t, t^b, \mathbf{t}_\ell) + \frac{\theta}{\theta_w}\psi_w(t, t^b, \mathbf{t}_\ell) \tag{2.4}$$

Again, the proof is given in the Appendix.

2.5.2 Reliability of a Hybrid k-out-of-n:G System

After resolving the lifetime reliability of a single component, we then move to study the lifetime reliability of the entire system, calculating its mean time to failure $MTTF^{sys}$. Let $P^{sys}_{j,i}(t)$ be the probability that the hybrid redundant system has j active components, and i good spare components at time t. As a functioning hybrid k-out-of-n:G system may have m active components with no more than $(n-m)$ good spare ones, or no less than k active components without good spare ones, the system reliability $P^{sys}(t)$ can be expressed by two summations:

$$P^{sys}(t) = \sum_{i=0}^{n-m} P^{sys}_{m,i}(t) + \sum_{j=k}^{m-1} P^{sys}_{j,0}(t). \tag{2.5}$$

Hence, the mean time to failure of the entire system

$$MTTF^{sys} = \int_0^\infty P^{sys}(t)\mathrm{d}t \tag{2.6}$$

can be written as

$$MTTF^{sys} = \int_0^{\infty} \Big[\sum_{i=0}^{n-m} P_{m,i}^{sys}(t) + \sum_{j=k}^{m-1} P_{j,0}^{sys}(t) \Big] \mathrm{d}t$$
$$= \int_0^{\infty} \sum_{i=0}^{n-m} P_{m,i}^{sys}(t)\mathrm{d}t + \int_0^{\infty} \sum_{j=k}^{m-1} P_{j,0}^{sys}(t)\mathrm{d}t. \quad (2.7)$$

For the sake of simplicity, let $P_{n-\ell}^{sys}(t)$ be the probability of a system containing $(n-\ell)$ good components, including both active components, and good spares. Equation (2.7) can therefore be rewritten as

$$MTTF^{sys} = \int_0^{\infty} \sum_{\ell=0}^{n-k} P_{n-\ell}^{sys}(t)\mathrm{d}t \quad (2.8)$$

$P_n^{sys}(t)$ is simply the probability that the system has no failures up to time t. As the $(n-m)$ spare components have zero failure rate, $P_n^{sys}(t)$ is the probability that all m active components do not fail from t_0 to t, i.e.,

$$P_n^{sys}(t) = R^m(t, t_0|\mathbf{t}_0) \quad (2.9)$$

If a failure occurs at time t_1, a spare component converts into active mode with a very short reconfiguration time (when compared to the system's lifetime). Before the next failure, this system consists of $(m-1)$ active components with birth time t_0, and one with birth time t_1. Thus, the conditional probability that the system contains $(n-1)$ good components at time t ($t > t_1$), given a component with birth time t_0 fails at t_1, is given by

$$P_{n-1}^{sys}(t|\mathbf{t}_1;(0)) = R^{m-1}(t, t_0|\mathbf{t}_1) \cdot R(t, t_1|\mathbf{t}_1) \quad (2.10)$$

Here, vector (0) in the notation $P_{n-1}^{sys}(t|\mathbf{t}_1;(0))$ represents that the failure component has birth time t_0. Related notations will be formally introduced later.

Because the probability that an indicated component with birth time t_0 fails at time t_1 is $\frac{\mathrm{d}}{\mathrm{d}t}(1-R(t,t_0|\mathbf{t}_0))\Big|_{t=t_1}$, and there are m such kind of components, the probability that any one of them fails at t_1 is

$$g_n^{sys}(t_1) = m\frac{\mathrm{d}}{\mathrm{d}t}(1-R(t,t_0|\mathbf{t}_0))\Big|_{t=t_1} \tag{2.11}$$

The event that the system experiences exactly one failure up to t is a union of a set of continuous elementary events in which a failure occurred in an infinitesimal interval $\mathrm{d}t_1$ at time t_1, and the probability for this is $P_{n-1}^{sys}(t|\mathbf{t}_1;(0))g_n^{sys}(t_1)\mathrm{d}t_1$. By the theorem of total probability, the unconditional probability can be obtained by integration over t_1, i.e.,

$$\begin{aligned} P_{n-1}^{sys}(t) = &\int_0^t R^{m-1}(t,t_0|\mathbf{t}_1)\cdot R(t,t_1|\mathbf{t}_1)\cdot \\ &\cdot m\frac{\mathrm{d}}{\mathrm{d}t}(1-R(t,t_0|\mathbf{t}_0))\Big|_{t=t_1}\mathrm{d}t_1 \end{aligned} \tag{2.12}$$

When $\ell \geq 2$, because there might be more than one possible case of failures, the expression of $P_{n-\ell}^{sys}(t)$ becomes more complex. To estimate it, we use two $1\times\ell$ row vectors $\mathbf{t}_\ell = (t_1,t_2,\cdots,t_\ell)$, and $\mathbf{x}_\ell = (x_1,x_2,\cdots,x_\ell)$ to capture the dominant characteristic of ℓ failures. t_i represents the occurrence time of the i^{th} failure. x_i indicates the birth time index of the i^{th} failure component, meaning that, if the i^{th} failure component has birth time t_r, the corresponding birth time index x_i is r. In general, there are 2ℓ elements

$$\begin{Bmatrix} t_1 & t_2 & \cdots & t_\ell \\ x_1 & x_2 & \cdots & x_\ell \end{Bmatrix} \tag{2.13}$$

each column corresponding to a failure. The occurrence time satisfies the constraints that

$$t_1 < t_2 < \cdots < t_\ell \tag{2.14}$$

Because the i^{th} failure component must convert into active mode before t_i, its birth time $t^b \in$ set $\{t_0, t_1, \cdots, t_{i-1}\}$. On the other hand, all components start to serve as active ones on or before t_{n-m}. Therefore, the birth time indices satisfy the constraints that

$$x_i = 0, 1, \cdots, \min(i-1, n-m) \tag{2.15}$$

The quantity of i in the first j elements of vector $\mathbf{x}_\ell$ is denoted as $\pi_{i,j}$. Because only one component turns into the active mode after each failure, there is essentially only one component with birth time t_i in the entire system if $i \neq 0$. In addition, the system contains no more than m components with birth time t_0. Hence, the birth time indices should also satisfy

$$\forall x_i, x_j \neq 0 : x_i \neq x_j \qquad \text{and} \qquad \pi_{0,\ell} \leq m \tag{2.16}$$

For a possible $\mathbf{x}_\ell$, let $\mathbf{x}_\ell^{(r)} = (x_1, x_2, \cdots, x_r)$ be the r-dimensional subvector of $\mathbf{x}_\ell$. Similarly, let $\mathbf{t}_\ell^{(r)} = (t_1, t_2, \cdots, t_r)$ be the corresponding r-dimensional subvector of $\mathbf{t}_\ell$. That is, for any possible case of failures, its first r failures can always be described by $\mathbf{x}_\ell^{(r)}$, and $\mathbf{t}_\ell^{(r)}$. Further, we define

$$f(t_r, t^b | \mathbf{t}_\ell^{(r-1)}) \equiv \frac{\mathrm{d}}{\mathrm{d}t}(1 - R(t, t^b | \mathbf{t}_\ell^{(r-1)}))\Big|_{t=t_r} \tag{2.17}$$

We therefore express the probability that the system contains $(n-\ell)$ good components having ℓ failures whose characteristics can be described by $\mathbf{t}_\ell$, and $\mathbf{x}_\ell$ as

$$P^{sys}_{n-\ell}(t,\mathbf{t}_\ell;\mathbf{x}_\ell) = P^{sys}_{n-\ell}(t|\mathbf{t}_\ell;\mathbf{x}_\ell)g^{sys}_n(t_1;x_1)\cdot \cdot g^{sys}_{n-1}(t_2;x_2|\mathbf{t}^{(1)}_\ell;\mathbf{x}^{(1)}_\ell)\cdots \cdots g^{sys}_{n-\ell+1}(t_\ell;x_\ell|\mathbf{t}^{(\ell-1)}_\ell;\mathbf{x}^{(\ell-1)}_\ell) \quad (2.18)$$

where, $P^{sys}_{n-\ell}(t|\mathbf{t}_\ell;\mathbf{x}_\ell)$ is the conditional probability that the system contains $(n-\ell)$ good components at time t, given the past ℓ failures described by $\mathbf{t}_\ell$, and $\mathbf{x}_\ell$. $g^{sys}_{n-r+1}(t_r;x_r|\mathbf{t}^{(r-1)}_\ell;\mathbf{x}^{(r-1)}_\ell)$ denotes the probability that, in the system containing $(n-r+1)$ good components, a component with birth time t_{x_r} fails at time t_r given the past $(r-1)$ failures can be described by $\mathbf{t}^{(r-1)}_\ell$ and $\mathbf{x}^{(r-1)}_\ell$.

After ℓ failures, the number of good components with birth time t_0 is $(m-\pi_{0,\ell})$; and that with birth time t_i $(i\neq 0)$ is $(1-\pi_{i,\ell})$. Consequently, the conditional probability $P^{sys}_{n-\ell}(t|\mathbf{t}_\ell;\mathbf{x}_\ell)$ can be computed as

$$P^{sys}_{n-\ell}(t|\mathbf{t}_\ell;\mathbf{x}_\ell) = R^{m-\pi_{0,\ell}}(t,t_0|\mathbf{t}_\ell)\cdot \prod_{i=1}^{\min(n-m,\ell)} R^{1-\pi_{i,\ell}}(t,t_i|\mathbf{t}_\ell) \quad (2.19)$$

As for the computation of $g^{sys}_{n-r+1}(t_r;x_r|\mathbf{t}^{(r-1)}_\ell;\mathbf{x}^{(r-1)}_\ell)$, we consider two cases: (i) the r^{th} failure component has birth time t_0, or (ii) it has birth time t_i $(i\neq 0)$. For the first case, the probability that an indicated component with birth time t_0 fails at time t_r is $f(t_r,t_0|\mathbf{t}^{(r-1)}_\ell)$, and there are $(m-\pi_{0,r-1})$ surviving components with birth time t_0 in such a system. For the second case, there is only one good component with birth time t_{x_r}, and the probability for its failure at t_r is $f(t_r,t_{x_r}|\mathbf{t}^{(r-1)}_\ell)$. Therefore, we have

$$g^{sys}_{n-r+1}(t_r;x_r|\mathbf{t}^{(r-1)}_\ell;\mathbf{x}^{(r-1)}_\ell) = \begin{cases} (m-\pi_{0,r-1})\cdot \\ \cdot f(t_r,t_0|\mathbf{t}^{(r-1)}_\ell),\ x_r=0 \\ f(t_r,t_{x_r}|\mathbf{t}^{(r-1)}_\ell),\ \text{otherwise} \end{cases} \quad (2.20)$$

After analyzing a single failure case, we can now compute $P_{n-\ell}^{sys}(t)$ for all possible cases. Denote the set of all possible $\mathbf{x}_\ell$ as $\mathcal{X}_\ell$. Because $t_0 < t_1 < \cdots < t_\ell < t$, we have

$$P_{n-\ell}^{sys}(t) = \int_0^t \mathrm{d}t_1 \int_{t_1}^t \mathrm{d}t_2 \int_{t_2}^t \mathrm{d}t_3 \cdots \int_{t_{\ell-2}}^t \mathrm{d}t_{\ell-1} \int_{t_{\ell-1}}^t \mathrm{d}t_\ell \sum_{\mathbf{x}_\ell \in \mathcal{X}_\ell} P_{n-\ell}^{sys}(t, \mathbf{t}_\ell; \mathbf{x}_\ell) \tag{2.21}$$

Interchanging the integration limits yields

$$P_{n-\ell}^{sys}(t) = \int_0^t \mathrm{d}t_\ell \int_0^{t_\ell} \mathrm{d}t_{\ell-1} \int_0^{t_{\ell-1}} \mathrm{d}t_{\ell-2} \cdots \int_0^{t_3} \mathrm{d}t_2 \int_0^{t_2} \mathrm{d}t_1 \sum_{\mathbf{x}_\ell \in \mathcal{X}_\ell} P_{n-\ell}^{sys}(t, \mathbf{t}_\ell; \mathbf{x}_\ell) \tag{2.22}$$

2.6 Special Cases

In this section, we simplify the proposed models under the assumption that the system is a gracefully degrading one (Case I), or a standby redundant one (Case II) respectively, for verification purpose. Further, we assume constant failure rate in both cases, resulting in exactly the same results as previous work. We also present a special case in Case III to demonstrate the detailed analytical procedure by using the proposed approach.

2.6.1 Case I: Gracefully Degrading System

When $n = m$, the system discussed above becomes a load-sharing k-out-of-n:G gracefully degrading system. In this case, there is only one possible failure case for any ℓ, i.e.,

$$x_\ell = \{(\underbrace{0,\cdots,0}_{\ell})\}$$

Thus, dropping $\mathbf{x}_\ell$ from notations, and rewriting (2.22), (2.18)-(2.20), yields

$$P^{sys}_{n-\ell}(t) = \int_0^t \mathrm{d}t_\ell \int_0^{t_\ell} \mathrm{d}t_{\ell-1} \int_0^{t_{\ell-1}} \mathrm{d}t_{\ell-2} \cdots \int_0^{t_3} \mathrm{d}t_2 \int_0^{t_2} \mathrm{d}t_1 P^{sys}_{n-\ell}(t, \mathbf{t}_\ell)$$

where,

$$\begin{aligned} P^{sys}_{n-\ell}(t, \mathbf{t}_\ell) &= P^{sys}_{n-\ell}(t|\mathbf{t}_\ell) g^{sys}_n(t_1) g^{sys}_{n-1}(t_2|\mathbf{t}^{(1)}_\ell) \cdots \\ &\quad \cdots g^{sys}_{n-\ell+1}(t_\ell|\mathbf{t}^{(\ell-1)}_\ell) \end{aligned} \tag{2.23}$$

$$P^{sys}_{n-\ell}(t|\mathbf{t}_\ell) = R^{n-\ell}(t, t_0|\mathbf{t}_\ell)$$

$$g^{sys}_{n-r+1}(t_r|\mathbf{t}^{(r-1)}_\ell) = (n-r+1) f(t_r, t_0|\mathbf{t}^{(r-1)}_\ell)$$

If we further assume that all components have the same constant failure rate in both processing and wait states, then

$$\mathcal{R}(t) = e^{-\frac{t}{\theta}}, \qquad R_p(t) = R_w(t) = e^{-\frac{t}{\theta_p}}$$

and the computation is greatly simplified. For the ease of expression, let $\eta = \frac{1}{\theta}$. By Theorems 1, and 2, we have

$$\begin{aligned} R(t, t_0|\mathbf{t}_\ell) &= \mathcal{R}(\psi(t, t_0, \mathbf{t}_\ell)) \\ &= \mathcal{R}(\frac{\theta}{\theta_p}\psi_p(t, t_0, \mathbf{t}_\ell) + \frac{\theta}{\theta_w}\psi_w(t, t_0, \mathbf{t}_\ell)) \\ &= \mathcal{R}(\frac{\theta}{\theta_p} \cdot (\psi_p(t, t_0, \mathbf{t}_\ell) + \psi_w(t, t_0, \mathbf{t}_\ell))) \\ &= \mathcal{R}(\frac{\theta}{\theta_p} \cdot t) = e^{-\eta t} \end{aligned}$$

Thus, by (2.17),

$$f(t_r, t_0|\mathbf{t}_\ell^{(r-1)}) = \frac{\mathrm{d}}{\mathrm{d}t}(1-e^{-\eta t})\Big|_{t=t_r} = \eta e^{-\eta t_r}$$

Substituting them into (2.23), we obtain

$$P_{n-\ell}^{sys}(t) = e^{-(n-\ell)\eta t}\binom{n}{\ell}(1-e^{-\eta t})^\ell$$

With 2.8), we obtain

$$MTTF^{sys} = \int_0^\infty \sum_{\ell=0}^{n-k} P_{n-\ell}^{sys}(t)\mathrm{d}t = \frac{1}{\eta}\sum_{\ell=k}^{n}\frac{1}{\ell}$$

This is the same results shown in previous works [10].

2.6.2 Case II: Standby Redundant System

When $k = m$, this system becomes a load-sharing m-out-of-n:G standby redundant system. In this case, the number of failure components ℓ in a functioning system cannot exceed $(n-m)$; otherwise the system fails. Therefore, a surviving component's cumulative time in either processing or wait state only depends on its birth time t^b, i.e.,

$$\psi_p(t, t^b, \mathbf{t}_\ell) \approx \frac{\lambda}{m\mu}(t-t^b)$$

$$\psi_w(t, t^b, \mathbf{t}_\ell) \approx (1-\frac{\lambda}{m\mu})(t-t^b)$$

Again, if we further assume the failure rate to be constant, and $\theta_p = \theta_w = \frac{1}{\eta}$, we obtain

$$R(t, t^b|\mathbf{t}_\ell) = e^{-\eta(t-t^b)}$$

$$f(t_r, t^b|\mathbf{t}_\ell^{(r-1)}) = \eta e^{-\eta(t_r-t^b)}.$$

Thus, the system reliability at time t becomes

$$P^{sys}(t) = e^{-m\eta t}\sum_{\ell=0}^{n-m}\frac{(m\eta t)^\ell}{\ell!},$$

and mean time to failure of this system is

$$MTTF^{sys} = \frac{n-m+1}{m\eta}$$

which is the same result as that in [67].

2.6.3 Case III: 1-out-of-3:G System with m=2

In this system, $k = 1$, $m = 2$, and $n = 3$. By (2.8), it is necessary to determine $P_{3-\ell}^{sys}(t)$ $(0 \leq \ell \leq 2)$ to achieve $MTTF^{sys}$, where $P_{3-\ell}^{sys}(t)$ can be defined in two equivalent ways: first, the probability that the system contains $(3-\ell)$ good components at time t; and second, the probability that exactly ℓ failures have happened in the system up to time t.

Initially, two components are in the active mode, with the remaining one in the spare mode. The active components share the workload of the entire system. The traffic intensity of each component is $\frac{\lambda}{2\mu}$. If no failures occur in the system up to time t, the cumulative time of an active component remaining in the processing, and wait state are $\frac{\lambda}{2\mu}t$, and $(1-\frac{\lambda}{2\mu})t$ respectively. By Theorem 2, the probability that an active component survives at time t is given by

$$R(t,t_0|\mathbf{t}_0) = \mathcal{R}(\psi(t,t_0,\mathbf{t}_0)) = \mathcal{R}(\frac{\theta}{\theta_p}\cdot\frac{\lambda}{2\mu}t + \frac{\theta}{\theta_w}\cdot(1-\frac{\lambda}{2\mu})t)$$

Substituting this reliability function into (2.9) yields

$$P_3^{sys}(t) = R^2(t,t_0|\mathbf{t}_0) = \mathcal{R}^2(\frac{\theta}{\theta_p}\cdot\frac{\lambda}{2\mu}t + \frac{\theta}{\theta_w}\cdot(1-\frac{\lambda}{2\mu})t). \tag{2.24}$$

After the first failure, which may occur on either active component with the same failure rate, the spare component is activated; and its traffic intensity is $\frac{\lambda}{2\mu}$. The traffic intensity of the surviving active component remains at the past level. There is only one possible case of failure, that is, $\mathbf{t}_1 = (t_1)$, and $\mathbf{x}_1 = (0)$. Suppose exactly one failure occurs up to time t, and the occurrence time is denoted as t_1.

Then by Theorem 1, the cumulative time of the component with birth time t_0 in the processing, and wait state are $\frac{\lambda}{2\mu}t$, and $(1-\frac{\lambda}{2\mu})t$ respectively; while those time intervals of the component with birth time t_1 are $\frac{\lambda}{2\mu}(t-t_1)$, and $(1-\frac{\lambda}{2\mu})(t-t_1)$ respectively. Using Theorem 2, and substituting into (2.12) yields

$$P_2^{sys}(t) = \int_0^t R(t,t_0|\mathbf{t}_1)\cdot R(t,t_1|\mathbf{t}_1)\cdot \cdot 2\cdot\frac{\mathrm{d}}{\mathrm{d}t}(1-R(t,t_0|\mathbf{t}_0))\Big|_{t=t_1}\mathrm{d}t_1 \tag{2.25}$$

where,

$$R(t,t_0|\mathbf{t}_1) = \mathcal{R}(\frac{\theta}{\theta_p}\cdot\frac{\lambda}{2\mu}t+\frac{\theta}{\theta_w}\cdot(1-\frac{\lambda}{2\mu})t),$$
$$R(t,t_1|\mathbf{t}_1) = \mathcal{R}(\frac{\theta}{\theta_p}\cdot\frac{\lambda}{2\mu}(t-t_1)+\frac{\theta}{\theta_w}\cdot(1-\frac{\lambda}{2\mu})(t-t_1)),$$

After the first failure, two components in the system have different ages, and hence different failure rates. When the system has experienced two failures up to t, there is only one good component left in the system at time t. This component's traffic intensity from t_2 to t is $\frac{\lambda}{\mu}$. By using vector $\mathbf{t}_\ell$, and vector $\mathbf{x}_\ell$ to describe all possible cases of two failures, we obtain two cases:

$$\begin{Bmatrix} t_1 & t_2 \\ 0 & 0 \end{Bmatrix} \quad \text{and} \quad \begin{Bmatrix} t_1 & t_2 \\ 0 & 1 \end{Bmatrix}$$

For the first case, the number of surviving components with birth time t_0 is $(m-\pi_{0,2}) = 0$, and that with birth time t_1 is $(1-\pi_{1,2}) = 1$, meaning that the only good component left after two failures has birth time t_1. This component's cumulative time in processing, and wait states are $\psi_p(t,t_1,\mathbf{t}_2) = \frac{\lambda}{\mu}t-\frac{\lambda}{2\mu}t_1-\frac{\lambda}{2\mu}t_2$, and $\psi_w(t,t_1,\mathbf{t}_2) = (1-\frac{\lambda}{\mu})t-(1-\frac{\lambda}{2\mu})t_1+\frac{\lambda}{2\mu}t_2$ respectively. Subvector $\mathbf{x}_2^{(1)} = (0)$. By (2.19), and (2.20), we obtain

$$P_1^{sys}(t|\mathbf{t}_2;\mathbf{x}_2) = R(t,t_1|\mathbf{t}_2)$$
$$g_3^{sys}(t_1;x_1) = 2f(t_1,t_0|\mathbf{t}_2^{(0)})$$
$$g_2^{sys}(t_2;x_2|\mathbf{t}_2^{(1)};\mathbf{x}_2^{(1)}) = f(t_2,t_0|\mathbf{t}_2^{(1)})$$

For the second case, the remaining component has birth time t_0, and subvector $\mathbf{x}_2^{(1)}$ is also (0). Similar to the analysis of the first case, we obtain $\psi_p(t,t_0,\mathbf{t}_2) = \frac{\lambda}{\mu}t - \frac{\lambda}{2\mu}t_2$, $\psi_w(t,t_0,\mathbf{t}_2) = (1-\frac{\lambda}{\mu})t + \frac{\lambda}{2\mu}t_2$, and

$$P_1^{sys}(t|\mathbf{t}_2;\mathbf{x}_2) = R(t,t_0|\mathbf{t}_2)$$
$$g_3^{sys}(t_1;x_1) = 2f(t_1,t_0|\mathbf{t}_2^{(0)})$$
$$g_2^{sys}(t_2;x_2|\mathbf{t}_2^{(1)};\mathbf{x}_2^{(1)}) = f(t_2,t_1|\mathbf{t}_2^{(1)})$$

With (2.22), we have

$$\begin{aligned} P_1^{sys}(t) &= \int_0^t \mathrm{d}t_2 \int_0^{t_2} \mathrm{d}t_1 \sum_{\mathbf{x}_\ell \in X_\ell} P_1^{sys}(t,\mathbf{t}_2;\mathbf{x}_2) \\ &= \int_0^t \mathrm{d}t_2 \int_0^{t_2} \mathrm{d}t_1 \Big[R(t,t_1|\mathbf{t}_2)\cdot 2f(t_1,t_0|\mathbf{t}_2^{(0)})\cdot \\ &\quad \cdot f(t_2,t_0|\mathbf{t}_2^{(1)}) + R(t,t_0|\mathbf{t}_2)\cdot \\ &\quad \cdot 2f(t_1,t_0|\mathbf{t}_2^{(0)})\cdot f(t_2,t_1|\mathbf{t}_2^{(1)})\Big]. \end{aligned} \tag{2.26}$$

Combining (2.24), (2.25), and (2.26), we finally obtain the lifetime reliability, and mean time to failure of this system:

$$P^{sys}(t) = P_3^{sys}(t) + P_2^{sys}(t) + P_1^{sys}(t)$$

$$MTTF^{sys} = \int_0^{\infty} P^{sys}(t)\mathrm{d}t$$

2.7 Numerical Results

The relentless scaling of CMOS technology has enabled the integration of a great amount of embedded processor cores on a single silicon die. Because of their advantages in power-efficiency, and short time-to-market, such large-scale multi-core systems have received lots of attention from both industry [77, 103, 108], and academia [4, 17]. At the same time, the ever-increasing on-chip power density accelerates the aging effect caused by various failure mechanisms, making their lifetime reliability a serious concern [16, 95]. As a consequence, designers typically introduce on-chip redundant cores to make the product fault-tolerant (e.g., [77, 25]). In this section, we present numerical results obtained with Monte Carlo integration based on our closed-form expressions for the lifetime reliability analysis of multi-core systems with different redundancy schemes, and various workloads. In particular, we first analyze the system with the proposed modeling method to achieve the closed-form expressions, and then resort to the Monte Carlo method to numerically approximate the values of multiple integrals in the resulting expressions.

2.7.1 Experimental Setup

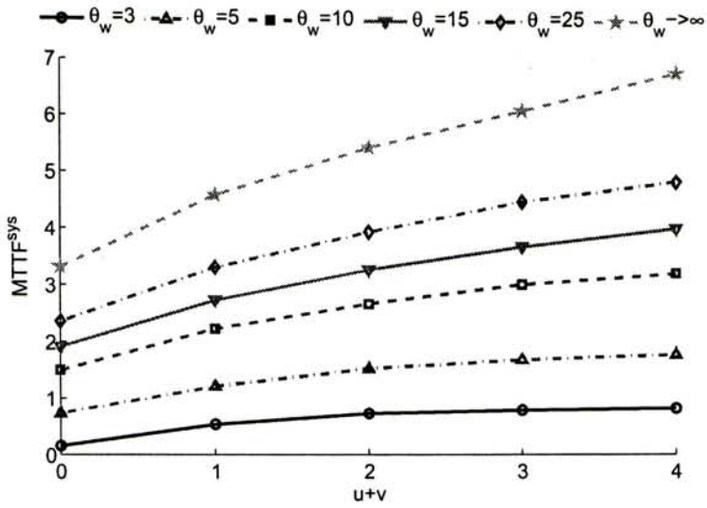
Two widely-used non-exponential lifetime distributions are assumed in the experiments: Weibull and linear failure rate. These reliability functions can be written as $\mathcal{R}(t) = e^{-(\frac{t}{\theta})^{\beta}}$, and $\mathcal{R}(t) = e^{-a\cdot(\frac{t}{\theta})-b\cdot(\frac{t}{\theta})^2}$, respectively. The scale parameters are different in the processing state (θ_p), and wait state (θ_w). Typically they are in units of years or hours. Clearly, θ_p is no more than θ_w. The property of the

$u+v$	Sojourn Time (years)					$MTTF^{sys}$
	0-Failure State	1-Failure State	2-Failure State	3-Failure State	4-Failure State	
0 + 0	0.2188	—	—	—	—	0.2188
1 + 0	0.2121	0.2188	—	—	—	0.4309
0 + 1	0.2188	0.2188	—	—	—	0.4376
2 + 0	0.2059	0.2121	0.2188	—	—	0.6368
1 + 1	0.2121	0.2121	0.2188	—	—	0.6430
0 + 2	0.2188	0.2188	0.2188	—	—	0.6564
3 + 0	0.2000	0.2059	0.2121	0.2188	—	0.8368
2 + 1	0.2059	0.2059	0.2121	0.2188	—	0.8427
1 + 2	0.2121	0.2121	0.2121	0.2188	—	0.8551
0 + 3	0.2188	0.2188	0.2188	0.2188	—	0.8752
4 + 0	0.1944	0.2000	0.2059	0.2121	0.2188	1.0312
3 + 1	0.2000	0.2000	0.2059	0.2121	0.2188	1.0368
2 + 2	0.2059	0.2059	0.2059	0.2121	0.2188	1.0486
1 + 3	0.2121	0.2121	0.2121	0.2121	0.2188	1.0672
0 + 4	0.2188	0.2188	0.2188	0.2188	0.2188	1.0940

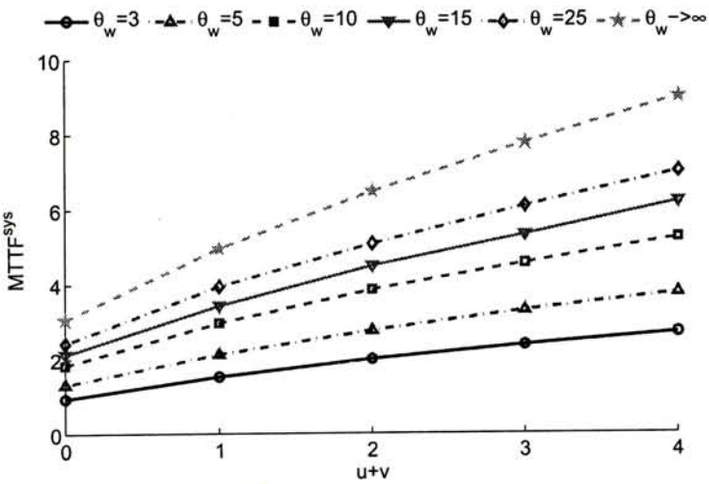
Table 2.1: Lifetime Reliability of Multi-core System with Constant Failure Rate.

Weibull distribution, whose failure rate function $h(t) = \frac{\beta}{\theta} \cdot (\frac{t}{\theta})^{\beta-1}$, highly depends on its shape parameter β. We set $\beta = 4$ in our experiment, implying an increasing failure rate with respect to time. The linear failure rate distribution has the hazard function $h(t) = \frac{a}{\theta} + \frac{2b}{\theta^2} \cdot t$, where $a, b \geq 0$. When $b = 0$, it reduces to an exponential distribution; when $a = 0$, it becomes a Rayleigh distribution. Different from the Weibull distribution, the linear failure rate distribution may have non-zero failure rate at $t = 0$. We set $a = 0.03$, $b = 0.15$ in our experiments.

The number of embedded cores in the multi-core system is set to be $(32 + u + v)$, meaning that the system has $(32 + u)$ active cores while the remaining v cores are put aside at time zero. If an active core is detected to be faulty, the system replaces it with a spare one until there are no spares in the system. Then the system enters its degrading phase until the number of good cores is less than



(a) Weibull Distribution



(b) Linear Failure Rate Distribution

Figure 2.3: Lifetime Enhancement of Multi-core System.

32. In other words, this system has parameters $n = 32 + u + v$, $m = 32 + u$, and $k = 32$. It becomes a load-sharing k-out-of-n:G gracefully degrading system when $v = 0$, and a standby redundant system when $u = 0$.

2.7.2 Experimental Results and Discussion

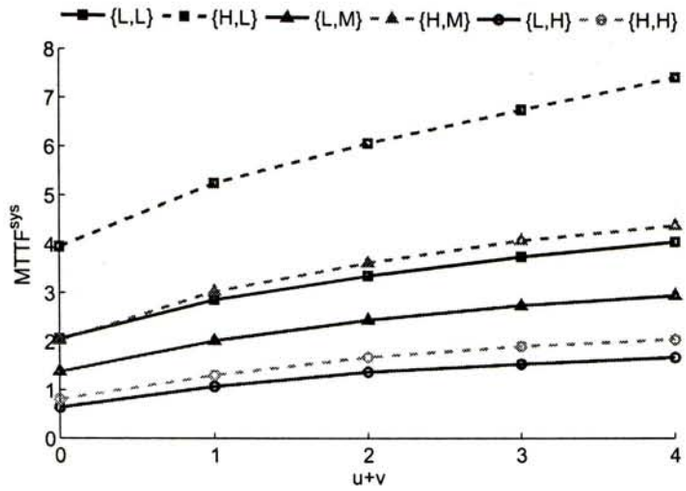
We discuss an issue that attracts attention: how much benefit can be expected from adding redundant cores into a multi-core system? As shown in Table 2.1, if we assume an exponential lifetime distribution, the sojourn time only depends on the number of active cores in the system, independent of the aging effect. With this assumption, we expect significant lifetime enhancement using redundant cores (around $(u+v)$ times extension), as shown in the last column of Table 2.1, where we set $\theta_w = \theta_p = 7$. For instance, the system lifetime increases from 0.2188 to around 0.64 by employing two redundant cores. Clearly, this result does not conform to our common sense. In practice, IC products experience increasing failure rates over their life cycles. In this sense, a Weibull or linear failure rate distribution could be a better approximation of such wear-out effect, and bring us more reasonable results.

Fig. 2.3 shows the lifetime enhancement achieved by redundant cores with Weibull and linear failure rate distributions for $\theta_p = 3$. The same quantity of redundant cores could have different redundant schemes, and hence result in $MTTF^{sys}$ deviations. In these figures, we simply plot the maximum $MTTF^{sys}$ achieved with the given $(u+v)$ because the area overhead depends on the core quantity only. Needless to say, the lifetime reliability of multi-core systems is enhanced with redundant cores at the cost of area overhead. At the same time, the lifetime improvement gradually slows down with the increase of $(u+v)$. For example, see the curve for the case with $\theta_w = 25$ in Fig. 2.7.1. The addition of first redundant core results in 26.66% lifetime extension; those of the second, third, and fourth one lead to 17.85%, 14.46%, and 9.34% extension, respectively. Consequently, designers need to set $(u+v)$ with an appropriate value to tradeoff area overhead with lifetime extension, rather than set $(u+v)$ as large as possible under the area overhead constraints.

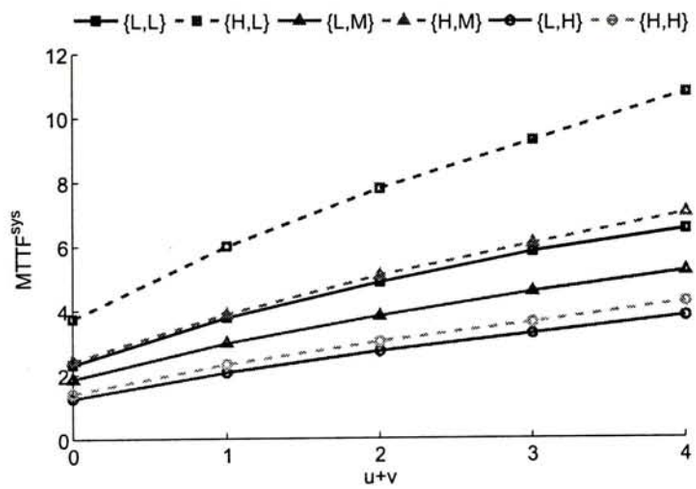
$u+v$	Sojourn Time (years)					$MTTF^{sys}$
	0-Failure State	1-Failure State	2-Failure State	3-Failure State	4-Failure State	
0+0	2.2039	—	—	—	—	2.2039
1+0	2.2153	0.5574	—	—	—	2.7727
0+1	2.2039	0.5617	—	—	—	2.7656
2+0	2.2260	0.5601	0.3593	—	—	3.1453
1+1	2.2153	0.5643	0.3581	—	—	3.1377
0+2	2.2039	0.5617	0.3558	—	—	3.1213
3+0	2.2359	0.5626	0.3642	0.2398	—	3.4025
2+1	2.2260	0.5667	0.3864	0.2578	—	3.4368
1+2	2.2153	0.5643	0.3613	0.2446	—	3.3855
0+3	2.2039	0.5617	0.3558	0.2375	—	3.3588
4+0	2.2452	0.5649	0.3633	0.2672	0.1555	3.5961
3+1	2.2359	0.5689	0.3765	0.2754	0.1652	3.6219
2+2	2.2260	0.5667	0.3663	0.2657	0.1750	3.5995
1+3	2.2153	0.5643	0.3613	0.2560	0.1480	3.5426
0+4	2.2039	0.5617	0.3558	0.2375	0.1069	3.4658

Table 2.2: Lifetime Reliability of Multi-core System with Non-Exponential Lifetime Distribution (Weibull).

We also need to place emphasis on the phenomenon that lifetime reliability highly depends on the scale parameters (i.e., θ_p, and θ_w) in the reliability functions. There are two extreme cases. The first is $\theta_p = \theta_w$, meaning that there is no difference between wait state and processing state in terms of the reliability function. It is essentially the so-called hot standby scheme. The second is $\theta_w \to \infty$, implying that an embedded core in the wait state is a cold standby component, and cannot fail. Taking the Linear failure rate distribution as an example, $MTTF^{sys}$ values in these two cases are 6.4324, and 1.6690 with four redundant cores, as shown in Fig. 2.7.1. Due to this huge gap, we advocate to reexamine the conclusions made under the hot or cold standby assumption, and deal carefully with the components in the wait state.



(a) Weibull Distribution



(b) Linear Failure Rate Distribution

Figure 2.4: Variation in Lifetime Reliability with Workload.

A closer observation for various redundant schemes is shown in Table 2.2, setting $\theta_p = 3$, $\theta_w = 10$, and $\frac{\lambda}{\mu} = 10$. Due to the increasing failure rate, the multi-core system contains no faulty cores in most of its lifetime, especially for systems suffering from severe wear-out effect. For example, consider the $(32+2+1)$ multi-core system. Its sojourn time in the 0-failure state is 2.2260 years, while the expected value of its whole lifetime is 3.4368 years. From this perspective,

one core's failure may imply the entire system is old, and we cannot expect much residual useful lifetime. Another interesting observation is, given the number of redundant cores, the maximum $MTTF^{sys}$ could occur with a hybrid redundant scheme. For example, the multi-core system with four redundant cores achieves its maximum lifetime when $u = 3$, and $v = 1$.

Finally, we study the influence of workloads on the lifetime reliability, and plot the results for $\theta_p = 3$ in Fig. 2.4, where the values of θ_w and λ/μ are labeled as $\{A,B\}$ in the legend and $A = \{L(10),H(25)\}$, $B = \{L(5),M(10),H(20)\}$. As can be observed, the workload has significant influence on the lifetime reliability of multi-core systems, and should be paid much attention by designers. That is, with the increase of workload $\frac{\lambda}{\mu}$, the system lifetime is significantly shortened, yet the scale of decrease in $MTTF^{sys}$ is much smaller than that of the increase in workload. Consider $\theta_w = 10$, and $u+v = 4$ in Fig. 2.7.2 as an example. The $MTTF^{sys}$ values are 4.5808, 3.6219, and 2.4750 for $\frac{\lambda}{\mu} = 5$, 10, and 20, respectively. We attribute this phenomenon to the wear-out effect of warm standby.

2.8 Conclusion

In this work, we present a general closed-form expression for the lifetime reliability of load-sharing k-out-of-n:G hybrid redundant systems. The load assigned to the system is modeled using queueing theory. We integrate the various failure distributions for components in different operational states into our analytical model with their corresponding aging effect, which are then used to estimate the lifetime reliability of the entire system. Finally, the practical applicability of the proposed model is verified with several special cases, and numerical experiments.

2.9 Appendix

Proof of Theorem 1

Case I. $\ell \leq n-m$. In this case, a surviving component's birth time $t^b \in$ set $\{t_0, t_1, \cdots, t_\ell\}$. Before t^b, the component serves as a cold standby. After that, it alternates between the processing and wait states. Because the number of faulty components is no more than $(n-m)$, there are exactly m active components in the system from t^b to t. According to queueing theory, the utilization ratio of an $M/M/1$ queue is $\frac{\lambda_q}{\mu}$, where λ_q is the arrival rate of this queue. In an equally load-sharing system with m active components, $\lambda_q = \frac{\lambda}{m}$. Because the time scale of the system lifetime is usually much larger than that of task processing, the cumulative time in the processing state can be approximated as $\frac{\lambda}{m\mu}(t-t^b)$. In addition, because the component is in either a processing or wait state from t^b to t, the cumulative time in the wait state $\psi_w(t, t^b, \mathbf{t}_\ell) = (t-t^b) - \psi_p(t, t^b, \mathbf{t}_\ell) = (1-\frac{\lambda}{m\mu})(t-t^b)$.

Case II. $\ell > n-m$. It is important to note that the birth time of any component must be no later than t_{n-m}, because any surviving component at time t $(t > t_\ell)$ has been configured as active at or before t_{n-m}. Therefore, the birth time $t^b \in \{t_0, t_1, \cdots, t_{n-m}\}$. From t^b to t_{n-m+1}, there are m active components in the system. Thus, the utilization ratio in this period is $\frac{\lambda}{m\mu}$. Hence, a component's cumulative time in the processing state from t^b to t_{n-m+1} is $\frac{\lambda}{m\mu}(t_{n-m+1}-t^b)$. From t_j to t_{j+1}, $(n-m+1 \leq j \leq \ell-1)$, the system contains $(n-j)$ active components. By the same argument, the component's cumulative time in the processing state from t_j to t_{j+1} is $\frac{\lambda}{(n-j)\mu}(t_{j+1}-t_j)$. Similarly, from t_ℓ to t, it is $\frac{\lambda}{(n-\ell)\mu}(t-t_\ell)$. Summing all these $(\ell-n+m+1)$ terms up results in (2.1). As for $\psi_w(t, t^b, \mathbf{t}_\ell)$, we compute it by $(t-t^b) - \psi_p(t, t^b, \mathbf{t}_\ell)$, similar to the computation for Case I.

□

Proof of Theorem 2

As $[t^b, t]$ is a closed interval, we can partition it into d sub-intervals according to state transitions: $t^b = T_0 < T_1 < T_2 < \cdots < T_d = t$. To be specific, at any T_i $(1 \leq i \leq d-1)$ the component converts from processing mode to wait mode, or opposite.

The initial reliability of a component is given by

$$R(T_0) = R(t^b) = \mathcal{R}(0).$$

Then, for the first sub-interval $[T_0, T_1)$, suppose the component does not have tasks to process in this time interval. Then the reliability at time τ $(T_0 \leq \tau < T_1)$ is given by

$$R(\tau) = \mathcal{R}(\frac{\theta}{\theta_w} \cdot (\tau - T_0)).$$

By this equation, at the end of this sub-interval, we have

$$R(T_1^-) = \mathcal{R}(\frac{\theta}{\theta_w} \cdot (T_1 - T_0)).$$

Next, we analyze the second sub-interval. Using c to represent the accumulated aging effect in $[T_0, T_1)$, the reliability at τ $(T_1 \leq \tau < T_2)$ can be written as

$$R(\tau) = \mathcal{R}(\frac{\theta}{\theta_p} \cdot (\tau - T_1 + c)).$$

Therefore, at the beginning of this sub-interval,

$$R(T_1^+) = \mathcal{R}(\frac{\theta}{\theta_p} \cdot c).$$

We then compute c by the continuity of the reliability function, that is, the reliability function must satisfy the following constraints

$$\forall \ell = 1, 2, \cdots, d-1 : R(T_\ell^-) = R(T_\ell^+).$$

Thus, because of $R(T_1^-) = R(T_1^+)$, we obtain $c = \frac{(T_1 - T_0)\theta_p}{\theta_w}$; and hence

$$R(\tau) = \mathcal{R}(\frac{\theta}{\theta_p} \cdot (\tau - T_1 + \frac{(T_1 - T_0)\theta_p}{\theta_w})), \qquad T_1 \leq \tau < T_2.$$

This equation implies that, if a component stays in the processing state for $\frac{(T_1-T_0)\theta_p}{\theta_w}$, its age is the same as if it had stayed in the wait state for $(T_1 - T_0)$. After a simple derivation, this equation can be further rewritten as

$$R(\tau) = \mathcal{R}\left(\frac{\theta}{\theta_p} \cdot (\tau - T_1) + \frac{\theta}{\theta_w} \cdot (T_1 - T_0)\right).$$

So, at time T_2, we have

$$R(T_2) = \mathcal{R}\left(\frac{\theta}{\theta_p} \cdot (T_2 - T_1) + \frac{\theta}{\theta_w} \cdot (T_1 - T_0)\right).$$

By generalizing the above calculation steps, the lifetime reliability of a component at time t can be written as

$$R(t) = \mathcal{R}\left(\frac{\theta}{\theta_p} \cdot \sum_{i=1}^{\lfloor d/2 \rfloor} (T_{2i} - T_{2i-1}) + \frac{\theta}{\theta_w} \cdot \sum_{i=1}^{\lceil d/2 \rceil} (T_{2i-1} - T_{2i-2})\right).$$

By Theorem 1, $\sum_{i=1}^{\lfloor d/2 \rfloor} (T_{2i} - T_{2i-1})$, and $\sum_{i=1}^{\lceil d/2 \rceil} (T_{2i-1} - T_{2i-2})$ have been approximated as $\psi_p(t, t^b, \mathbf{t}_\ell)$, and $\psi_w(t, t^b, \mathbf{t}_\ell)$ respectively. Additionally, this conclusion is obviously independent of the component's starting state. Therefore, (2.3) holds.

□

□ End of chapter.

Part II

Simulation Framework

Chapter 3

AgeSim: A Simulation Framework

Part of content in this chapter is included in the proceedings of *IEEE/ACM Design, Automation, and Test in Europe (DATE)* 2010 [47].

3.1 Introduction

Advancements in semiconductor technology has brought with enhanced functionality and improved performance in every new generation. At the mean time, the associated ever-increasing power and temperature density makes the lifetime reliability a serious concern in the industry, especially for the state-of-the-art system-on-chips that contain one or more embedded processors [16, 41, 95]. While the failure mechanisms have been extensively studied at the circuit level [15, 23], the accurate analysis at system level is still not easy task. For example, some integrated circuit products that have been shipped to market eventually have very high failure rates within the warranty period [78, 94], exposing the difficulties and imperfection of design process.

Needless to say, designers need to make sure that their system meets the lifetime reliability requirement. To achieve this objective, when they make decisions

that might affect reliability at design stage the wear-out effect must be taken into account. For example, various dynamic power/thermal management policies have been proposed for saving power and/or reducing power density in thermal hot spots and they have gained wide acceptance in the industry (e.g., [12, 19, 92]). These policies apparently affect processors' lifetime reliability significantly because the latter is highly related to the operational temperature and supply voltage of circuits. We therefore need to decide which policies to include in the design and how to tune their parameters under lifetime reliability constraint. In addition, providing fault-tolerance capabilities on-chip by incorporating redundant circuitries is an effective way for lifetime reliability enhancement [8, 97]. How much redundancy is enough to ensure the system's service life is an important decision to make at design stage to achieve reliable yet low-cost designs. Moreover, for multi-processor SoCs, how do we allocate applications to processors has a significant impact on the stress upon them and different allocation strategies may lead to remarkably different mean time to failure (MTTF) of the system [52, 49, 53]. Hence, again, when designers decide their task allocation strategies, they need to take the lifetime reliability factor into account.

Since the stress on processors vary significantly at runtime with different workloads, making the right decisions for the above mentioned design issues is extremely difficult, if not impossible, without an accurate lifetime reliability simulation framework. Obviously, it is unacceptable to build an experimental system and trace all the reliability-related factors over its lifetime (in the range of years) and use them for simulation. How to design an efficient yet accurate lifetime reliability simulator is therefore also a quite challenging problem, and there is only limited work in the literature in this domain [29, 82]. For the sake of simplicity, [29, 82] assumed an exponential lifetime distribution for each failure mechanism. In other words, the failure rate of the circuit is assumed to be only dependent on

its instantaneous behavior (e.g., temperature and voltage), independent of its usage history. This assumption is apparently inaccurate: a typical wear-out failure mechanism will have increasing failure rate as the circuit ages even if the operational temperature and voltage remain the same [46, 50].

In this paper, we propose a novel aging-aware simulation framework for evaluating the lifetime reliability of processor-based SoCs, namely *AgeSim*. *AgeSim* can simulate failure mechanisms with arbitrary lifetime distributions and hence is able to take their aging effect into account, which results in more accurate simulation results. In addition, *AgeSim* does not require to trace the system's reliability-related factors over its entire lifetime. Instead, tracing the representative application flows running on embedded processors once is sufficient for our simulation without sacrificing its accuracy much.

The main contributions of our work include: We propose a so-called *aging rate* concept to "hide" the impact of the SoC's reliability-related usage strategies (e.g., various DPM policies, trigger mechanisms, and application flow characteristics) with a single value. We then present a mathematical proof on how to express reliability function with aging rate and on the upper bound of inaccuracy induced by this approximation. This novel concept enables us to simulate the representative workloads once instead of simulating the SoC's activities over its entire lifetime. This model is sequentially extended to multi-processor systems with redundancy. We also present a novel simulation flow that extracts the distributions of processors' activities when executing representative workloads, which facilitates us to obtain the system's performance, MTTF, and energy consumptions efficiently.

The remainder of this paper is organized as follows. In Section 3.2, we present preliminaries and motivation for this work. The proposed lifetime reliability simulation framework *AgeSim* is then introduced in Section 3.3. Next, Section 3.4 details the calculation of aging rate with the simulation results of representative

workloads and validates its accuracy. We then extend the proposed lifetime reliability model for MPSoCs with redundant processor cores in Section 3.5. Four case studies are conducted in Section 3.6 to demonstrate the flexibility and effectiveness of the proposed methodology. Finally, Section 3.7 concludes this paper.

3.2 Preliminaries and Motivation

There are many kinds of failure mechanisms that could result in permanent errors of ICs. The most representative ones are electromigration on the interconnects, TDDB in the gate oxides, thermal cycling, and NBTI on PMOS transistors [2]. These failure mechanisms have an increasingly adverse effect with technology scaling, and hence are serious concerns for the semiconductor industry. The soft errors that are caused by radiation effect, although also important, are not viewed as lifetime reliability threats because they do not fundamentally damage the circuit [76]. We therefore focus on the former in this work.

3.2.1 Prior Work on Lifetime Reliability Analysis of Processor-Based Systems

While the above failure mechanisms have been extensively studied at the circuit level historically, it is essential to investigate their impact at the system level when analyzing the lifetime reliability of processor-based systems. This is because, these failures are strongly related to the temperature and voltage applied to the circuit [2, 96], while the processor's temperature vary significantly at runtime with different workloads [52, 53]. In addition, today's electrical systems are essentially adaptive systems, which change their runtime behaviors for power/thermal reduction. To be specific, the DPM and/or DTM policies being widely used in the industry include thermal throttling [26], module shutdown [13], dynamic voltage and frequency

scaling (DVFS) [21], and task migration among processor cores [92]. All have significant impact on the stress upon the embedded processors and hence their failure rates, which makes the lifetime reliability analysis quite complex.

Srinivasan *et al.* [95] described a so-called RAMP model for lifetime reliability analysis for microprocessors and proposed to conduct dynamic reliability management (DRM) using this model. In this work, the authors assumed a uniform device density over the chip and an identical vulnerability of devices to failure mechanisms. Later, Shin *et al.* [91] introduced a structure-aware model that takes the vulnerability of basic structures of the microarchitecture (e.g., register files, latches and logic) to different failure mechanisms into account. Exponential distribution for failure mechanisms were assumed in [91, 95] (equivalently, there are *no* aging effect for failure mechanisms), which makes these models inherently inaccurate.

There were also some recent work on simulation-based lifetime reliability analysis, which can be used to evaluate different DPM policies [29, 82]. These simulators contain a power management unit, implementing DPM policies, and a reliability monitoring unit, which gathers reliability-related information in the system (e.g., temperature and voltage) and uses them to obtain instantaneous failure rates. Similar to [91, 95], failure mechanisms' aging effect were not considered and hence they lead to inaccurate simulation results.

With the more realistic non-exponential lifetime distributions, circuits' reliability at a specific time point t depends on both its current reliability-related factors (e.g., temperature) and its past aging effect [50]. That is, even if a processor experiences the same stress at two different time points, their failure rates are different. To achieve accurate simulation, one possible method is to trace the processors' temperature and its execution parameters that affects reliability (e.g., voltage and frequency) throughout the entire lifetime, compute the corresponding lifetime reliability sequence, and finally integrate it over time t to obtain MTTF. Let us

take the commonly-used Weibull distribution $\mathcal{R}(t) = \exp\left(-(\frac{t}{\alpha})^{\beta}\right)$ for describing reliability function [2] as an example, where the scale parameter α depends on reliability-factors that changes at runtime (including temperature T and processors' execution state s) and shape parameter β hides the reliability-related factors that do not vary with time (e.g., structural properties of the circuits). Here, $\beta > 1$, if the failure rate increases over time. Depending on the temperature and execution state, the time horizon can be divided into a series of intervals (say, d intervals). By using this method, denoting by $\alpha(T_j, s_j)$ the scale parameters in the j^{th} interval and $\Delta_j\tau$ the interval length, the reliability at the d^{th} interval can be computed by $\mathcal{R}(t) = \exp\left(-(\sum_{i=1}^{d-1}\frac{\Delta_i\tau}{\alpha(T_i,s_i)} + \frac{t-\sum_{i=1}^{d-1}\Delta_i\tau}{\alpha(T_d,s_d)})^{\beta}\right)$.

Recently, Karl *et al.* [63] considered general lifetime distribution for failure mechanisms, and proposed to conduct DVFS according to reliability budget. In this paper, to verify the effectiveness of their proposed DRM policy, the authors conducted a 10-year lifetime simulation in their experiments using the above method. They collected real workload data from desktop computers and fill the 10-year time with randomly selected 1-hour workloads. Within each 1-hour period, they used a single temperature value to calculate reliability. This ignorance of temperature variation within the period results in lack of accuracy for their simulation results. Using a fine-grained simulation can mitigate the accuracy problem, however, it would lead to unaffordable simulation time.

3.2.2 Motivation of This Work

For systems at design stage, unlike in [63], it is impossible to obtain real workload information and simulate over its entire service life. In fact, due to the time-consuming temperature simulation, we can only simulate the system's execution for a short period. Therefore, we are facing the following challenging problem: How to achieve efficient yet accurate lifetime reliability simulation with such lim-

ited information, when failure mechanisms follow arbitrary failure rate distributions?

In addition, incorporating redundant circuitries on-chip is an effective way for lifetime reliability enhancement. Prior work (e.g., [29]) models multi-processor systems as parallel-serial systems [66] and calculates the lifetime reliability of the entire system accordingly. Using this model, however, also leads to inaccurate analytical results as it assumes all processors experience the same aging effect before they fail. Let us consider a standby redundant multi-processor system as an example. In such system, certain processors are initially set as spares and they become active only when some other active ones fail. Apparently, at the time point that a spare processor become active, it has a much smaller failure rate when compared to those processors that have already functioned for a long period. This effect, however, cannot be captured in the parallel-serial model. Consequently, how to take the various aging effect of different processors in a multi-processor system when simulating its lifetime reliability is also a challenging problem.

The above challenges motivate the proposed simulation framework investigated in this paper.

3.3 The Proposed Framework

Different from previous work, we propose to trace the representative workloads running on embedded processors in a fine-grained manner and use them to analyze the system's lifetime. This is feasible because as long as the probability of the target system being in each execution state and the temperature distribution obtained by the proposed approach conform to that in the whole service life, it can be used to represent the usage strategy of the system. In other words, the recorded information in this time duration is consistent with the usage strategy of the entire lifetime. Thus, if we can find out a quantity Ω (namely *aging rate*) that is able

to capture the impact of the processor's usage strategy on its aging effect and at the same time it is *independent* of time t, we are able to evaluate the processors' reliability with arbitrary failure distribution at any time in its service life. Before describing how to calculate Ω in detail (see Section 3.4), let us present the overall lifetime reliability simulation framework in this section.

Our fine-grained simulator *AgeSim*, used to evaluate the influence of various usage strategies on processor-based SoCs, is composed of three closely-related parts: *power/thermal manager*, *power simulator*, and *temperature simulator*, forming a feedback control loop, as shown in Fig. 3.1. Here, *usage strategy* of a system includes its application flow characteristics (e.g., the distribution of application service time), power states, and trigger mechanism for state transitions. For systems containing more than one processor core, it also includes load-sharing strategy among multiple cores, and redundancy scheme (e.g., gracefully degrading system), if any. Note that, we mainly consider the lifetime reliability of processor cores in *AgeSim* as they typically experience the highest wear-out stress in the system when compared to other hardware resources (e.g., peripherals). If, however, the reliability of these components are also of concern, our simulator can be easily extended to include them in the simulation framework.

The *power/thermal manager* determines the execution state of processors in the next time step based on what have occurred in the current time step. It is viewed as a black box, whose inputs and outputs are clear but can be implemented in any proper manner (power state machine is one of the choices [82]). It is worth noting that if the target system is an MPSoC, this part should include an application scheduler, which determines the processor cores that are used to execute each application. The *power simulator* evaluates the power consumption of every component according to their execution states and current application. The *temperature simulator* then takes the power consumption values and the temperature

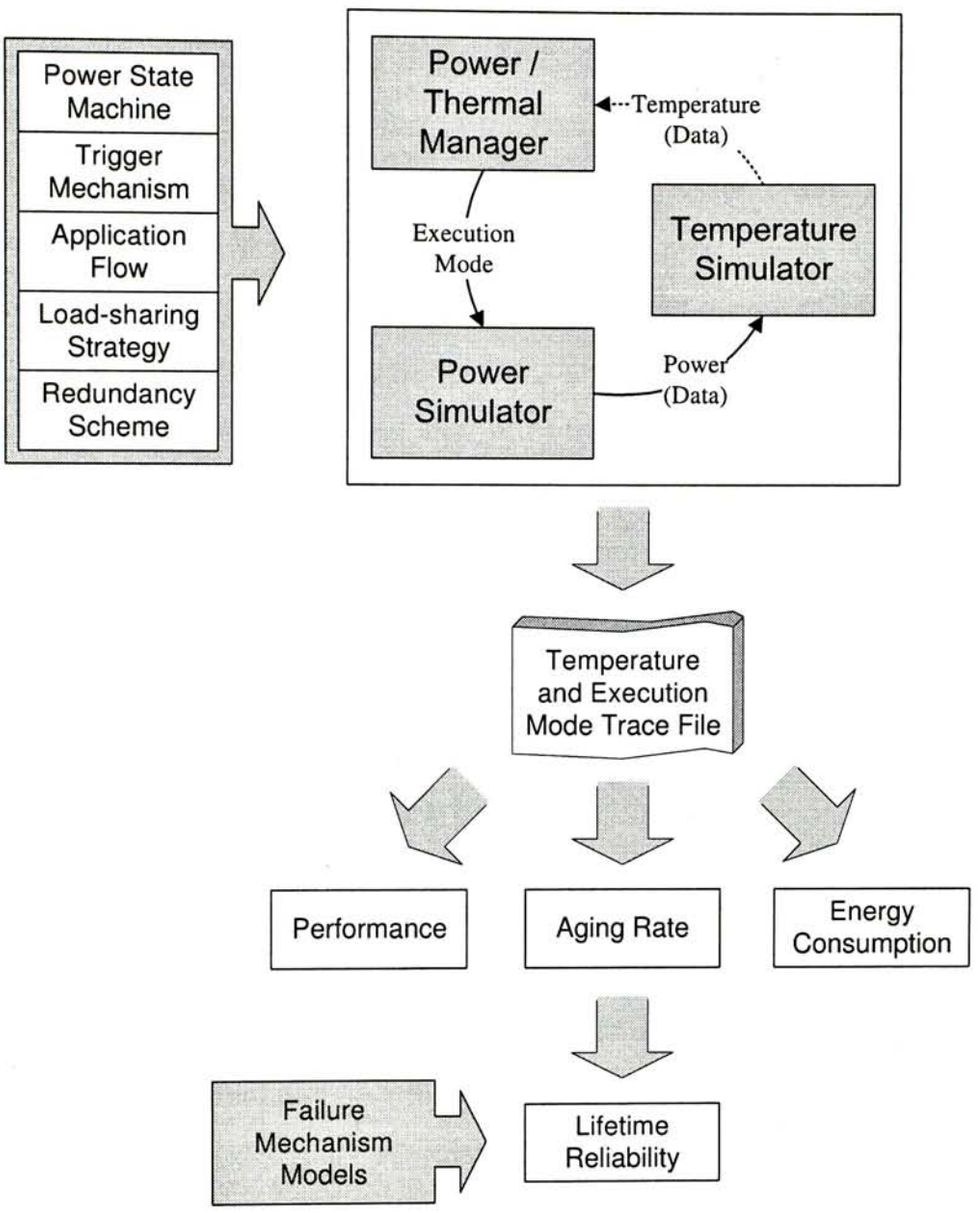


Figure 3.1: Lifetime Reliability Simulation Framework - *AgeSim*.

in the previous time step as inputs to obtain the temperature in the current time step. In *AgeSim*, we integrate HotSpot [92] into our simulator for accurate temperature computation. In our current implementation, temperature is used to trigger the execution state changes for a particular processor, if any. In case that the system's DPM/DTM policy requires other trigger mechanisms (e.g., processors' activity count [19]), they can be easily integrated into our simulation framework.

During simulation, we record the fine-grained temperature and execution state for every processor in each time step into a trace file. They are then used to compute the aging rate Ω and the expected lifetime of the system. At the same time, *AgeSim* also outputs the performance (e.g., mean response time) and energy consumption based on the traced information, which facilitates designers to evaluate their system from various aspects.

3.4 Aging Rate Calculation

According to earlier discussions, the key issue to achieve efficient yet accurate lifetime reliability simulation is to compute a time-independent aging rate Ω effectively with the limited traced information for representative workloads, so that we can express reliability as a function of Ω and t. This section shows how to achieve this objective using mathematical analysis. We tackle this problem by two steps: we first deduct a close-form lifetime reliability function with processors' time-varying operational states and temperature according to the reliability definition and property (Section 3.4.1), and then extract the time-independent aging rate parameter from this function (Section 3.4.2). The so-called representative work is then discussed in Section 3.4.3. Sequentially, the accuracy of the proposed model is validated in Section 3.4.4. Finally we discuss some miscellaneous issues in Section 3.4.5. It is important to note that, we target general failure distributions and we capture the aging impact of different workloads on processor cores, instead of

simply averaging out the aging-related parameters.

3.4.1 Lifetime Reliability Calculation

Existing circuit-level reliability models for hard failure mechanisms are not readily applicable to analyze processors' lifetime because their operational state and temperature vary significantly at run-time. We therefore propose a new high-level analytical model in this work.

Let $\mathcal{R}(t,\Theta)$ be a general failure distribution, where Θ represents the general scale parameter by which time t is divided. For instance, α is the scale parameter in Weibull distribution $\mathcal{R} = \exp\left(-(\frac{t}{\alpha})^{\beta}\right)$. As mentioned before, this parameter Θ is a function of temperature T. In addition, it also depends on several parameters that vary with processors' execution state, e.g., supply voltage and frequency for DVS-enabled processors. We therefore introduce another variable s to represent execution state and imply the state-related parameters. With these two variables, Θ can be written as $\theta(T,s)$. Without loss of generality, we assume there exists a set of possible execution states s and denote the set as $\mathcal{S}$.

As both T and s vary with respect to time t, we consider a finite sequence $0 = \tau_0 < \tau_1 < \cdots < \tau_d = t$ as a subdivision of time horizon $[0,t]$, including d intervals. Each interval j has the length $\Delta_j\tau = \tau_j - \tau_{j-1}$ and corresponds to a subsequence of temperature under a fixed operational state s_j. For all $\varepsilon > 0$, there exists $\delta > 0$ such that if the length of the largest interval $[\tau_i, \tau_{i+1}]$ $(i = \arg\max_j(\tau_{j+1} - \tau_j))$ is less than δ, the temperature variation within any interval is less than ε. Thus, for each interval j we can select an arbitrary temperature value from its temperature variation range for the entire interval, denoted as T_j. The corresponding scale parameter is therefore $\theta(T_j,s_j)$.

With the general failure distribution $\mathcal{R}(t,\Theta)$, since time t is divided by scale parameter, the reliability at time τ in the first interval (i.e., $\tau_0 \leq \tau < \tau_1$) can be

expressed as

$$R(\tau) = \mathcal{R}\left(\frac{\Theta}{\theta(T_1,s_1)} \cdot (\tau - \tau_0)\right), \qquad \tau_0 \leq \tau < \tau_1 \tag{3.1}$$

Then, we move to consider the j^{th} interval, where $j > 1$. Denoting by c_j the accumulated aging effect at the end of the j^{th} interval. Apparently, $c_0 = 0$. The lifetime reliability at time τ ($\tau_{j-1} \leq \tau < \tau_j$) is

$$R(\tau) = \mathcal{R}\left(c_{j-1} + \frac{\Theta}{\theta(T_j,s_j)} \cdot (\tau - \tau_{j-1})\right), \qquad \tau_{j-1} \leq \tau < \tau_j \tag{3.2}$$

With this equation, at the end of this interval (i.e., $\tau = \tau_j^-$)

$$R(\tau_j^-) = \mathcal{R}\left(c_{j-1} + \frac{\Theta}{\theta(T_j,s_j)} \cdot (\tau_j - \tau_{j-1})\right) \tag{3.3}$$

This time point is also the beginning of the $(j+1)^{\text{st}}$ interval. Therefore, we have

$$R(\tau_j^+) = \mathcal{R}(c_j) \tag{3.4}$$

By the continuity of reliability function, we have $R(\tau_j^-) = R(\tau_j^+)$ and we can express c_j as:

$$c_j = c_{j-1} + \frac{\Theta}{\theta(T_j,s_j)} \cdot (\tau_j - \tau_{j-1}) \tag{3.5}$$

With this expression, we obtain the reliability at time t

$$\begin{aligned} R(t) &= \mathcal{R}\left(\Theta \cdot \sum_{j=1}^{d} \frac{1}{\theta(T_j,s_j)} \cdot (\tau_j - \tau_{j-1})\right) \\ &= \mathcal{R}\left(\Theta \cdot \sum_{j=1}^{d} \frac{1}{\theta(T_j,s_j)} \cdot \Delta_j\tau\right) \end{aligned} \tag{3.6}$$

Taking limit as $\max_j \Delta_j\tau \to 0$ and $d \to \infty$, we have

$$R(t) = \mathcal{R}(\Theta \cdot \lim_{\substack{d\to\infty \\ \max_j \Delta_j\tau\to 0}} \sum_{j=1}^{d} \frac{1}{\theta(T_j, s_j)} \cdot \Delta_j\tau)) \tag{3.7}$$

In this equation, the state parameter s_j in any time interval j must belong to the set $\mathcal{S}$. We introduce an indicator function $\mathbb{1}_s^j$ to represent whether the state in the j^{th} interval is s. That is to say, if in the j^{th} time interval the processor is in execution state s, this function equals 1; otherwise, it is zero. With this notation, we can express the aging effect in various states separately and rewrite Eq. (3.7) as their summation, i.e.,

$$R(t) = \mathcal{R}(\Theta \cdot \sum_{s\in\mathcal{S}} \lim_{\substack{d\to\infty \\ \max_j \Delta_j\tau\to 0}} \sum_{j=1}^{d} \frac{\mathbb{1}_s^j}{\theta(T_j, s)} \cdot \Delta_j\tau)) \tag{3.8}$$

For integration, we define a filter function over time horizon $\mathbb{1}_s(\tau)$ such that it is one if τ falls into an interval with state s while zero otherwise. Therefore, we have

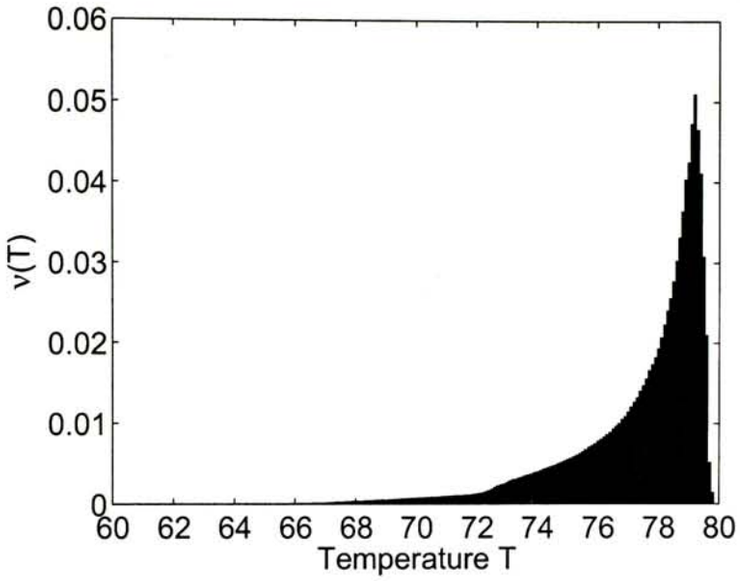
$$R(t) = \mathcal{R}(\Theta \cdot \sum_{s\in\mathcal{S}} \int_0^t \frac{\mathbb{1}_s(\tau)}{\theta(T, s)} \mathrm{d}\tau) \tag{3.9}$$

3.4.2 Aging Rate Extraction

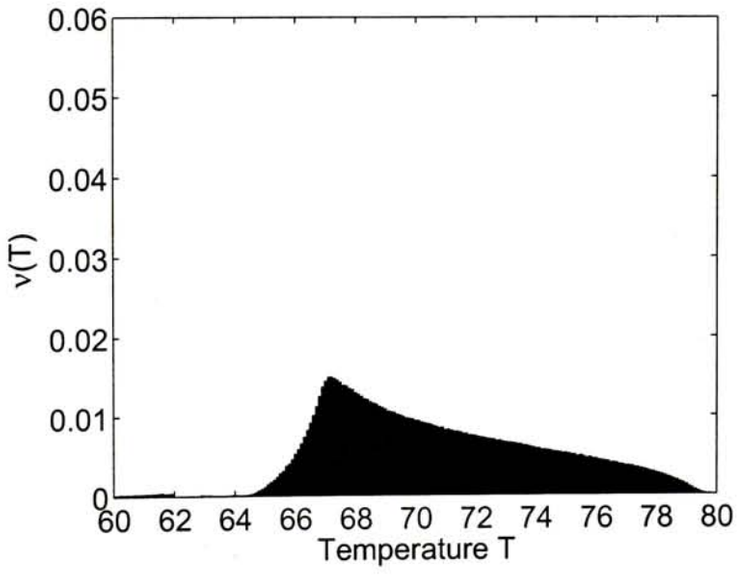
In the above, we have successfully express processors' reliability function using their high-level operational states and temperature values. However, as we integrate $\frac{1}{\theta(T,s)}$ over time in Eq. (3.9), we have not obtained the time-independent quantity yet. Fortunately, on the time horizon, the temperature T is a function of time τ. With this observation, we define $\psi(T,s)\mathrm{d}T$ as the accumulated time in state s in an infinitesimal temperature interval $\mathrm{d}T$ around T and use it to substitute $\mathrm{d}\tau$ in Eq. (3.9), leading to

$$R(t) = \mathcal{R}(\Theta \cdot \sum_{s\in\mathcal{S}} \int_0^\infty \frac{1}{\theta(T, s)} \cdot \psi(T, s)\mathrm{d}T) \tag{3.10}$$

The only term depending on time in Eq. (3.10) is $\psi(T,s)$. We use π_s to represent the probability a core being in state s, and $\nu(T,s)$ to indicate the conditional probability density function of a core having temperature T, given state



(a) Run State



(b) Idle State

Figure 3.2: Temperature Distribution Examples.

s. These values can be easily extracted via simulation. Fig. 3.2 shows two example temperature distributions extracted from the trace file for applying random task allocation on a 9-core processor (see Section 3.6.3 for detail). The accumulated time can therefore be expressed as the product of three quantities, i.e., $\psi(T,s) = \pi_s \cdot \nu(T,s) \cdot t$. Substituting it back to Eq. (3.10), we have

$$R(t) = \mathcal{R}\,(\Theta \cdot \sum_{s \in S} \int_0^{\infty} \frac{1}{\theta(T,s)} \cdot \pi_s \cdot \nu(T,s) \cdot t \mathrm{d}T) \tag{3.11}$$

We define

$$\Omega \equiv \sum_{s \in S} \int_0^{\infty} \frac{1}{\theta(T,s)} \cdot \pi_s \cdot \nu(T,s) \mathrm{d}T \tag{3.12}$$

Since the current time t is independent of all terms in the definition of Ω, we have successfully obtained a time-independent variable Ω, referred as *aging rate*, to express the reliability at time t as

$$R(t) = \mathcal{R}\,(\Theta \cdot \Omega \cdot t) \tag{3.13}$$

It is necessary to highlight that, since the characteristics of the representative workloads is consistent with that of entire lifetime, this equation is applicable for any time t in the entire service life. Consider Weibull distribution, that is, $\mathcal{R}\,(t) = \exp\big(-(\frac{t}{\Theta})^{\beta}\big)$. Substituting it into Eq. (3.13) yields

$$R(t) = \exp\big(-(\frac{\Theta \cdot \Omega \cdot t}{\Theta})^{\beta}\big) = \exp\big(-(\Omega \cdot t)^{\beta}\big) \tag{3.14}$$

Another point that should be highlighted is how to obtain the scale parameter $\theta(T_j, s_j)$ with its parameters T_j and s_j. As discussed before, existing circuit-level models for failure mechanism cannot be used directly, because they assume constant temperature and fixed aging-related parameters throughout the entire service life. For example, a widely-accepted lifetime reliability model for electromigration

is given by $MTTF_{EM} \propto (V_{dd} \times f)^{-2} e^{\frac{E_a}{kT}}$ [39], assuming fixed absolute temperature T, supply voltage V_{dd} and clock frequency f. Here, E_a and k are material related constant and the Boltzmann's constant respectively. However, since temperature T_j and operational state s_j that implies state-related parameters (e.g., supply voltage and frequency) at the j^{th} time interval can be assumed as constant parameters with our fine-grained tracing, existing failure models can be used to calculate $\theta(T_j, s_j)$ for this particular time interval.

If the target system contains one or more processors without redundancy (the number of processor cores is n), given core i's aging rate $\Omega_{s,i}$ in each state s, the system's service life can be simply computed by integrating the system reliability over time t, i.e.,

$$MTTF = \int_0^{\infty} \prod_{i=1}^{n} \mathcal{R}\left(\Theta \cdot \sum_{s \in S} \pi_{s,i} \cdot \Omega_{s,i} \cdot t\right) \mathrm{d}t \tag{3.15}$$

3.4.3 Discussion on Representative Workload

Recall that the representative workload is intuitively defined as the time duration in which the recorded information is consistent with the usage strategy of the entire service life. In this section, we transfer this description to mathematical language and theoretically elucidate the impact of this time duration on the estimation error of lifetime reliability.

For a system with certain usage strategy, if there exists a time interval with duration $\tau_d \ll MTTF$ such that the temperature distribution for each state remains the same for any time interval $[t, t+\tau_d]$, the workload in an arbitrary interval $[t, t+\tau_d]$ is a representative one.

The computation of mean time to failure with aging rate is essentially an approximation. Hence we are interested in the associated inaccuracy. The actually mean time to failure of the system that is approximated with the following equa-

tion in *AgeSim* reaches its extreme values in the cases that there is a reliability stress impulse with integral $\tau_d \cdot \Omega$ at the very beginning of every time duration τ_d (named case (a)) and that at the end of time duration (named case (b)). We denote by $MTTF_{\min}$ and $MTTF_{\max}$ the mean time to failure in these two cases because the real value must be no less than $MTTF_{\min}$ and cannot exceed $MTTF_{\max}$.

$$MTTF_{\text{approx}} = \int_0^{\infty} \mathcal{R}(\Theta \cdot \Omega \cdot t)\mathrm{d}t \tag{3.16}$$

As depicted in Fig. 3.3, the area inside the dotted curve is the system real mean time to failure while that in the solid rectangles show the extreme values, which can be expressed in the form

$$MTTF_{\min} = \sum_{i=1}^{\infty} \mathcal{R}(\Theta \cdot \Omega \cdot i \cdot \tau_d) \cdot \tau_d \tag{3.17}$$

and

$$MTTF_{\max} = \sum_{i=0}^{\infty} \mathcal{R}(\Theta \cdot \Omega \cdot i \cdot \tau_d) \cdot \tau_d \tag{3.18}$$

Clearly, the system lifetime reliability is overestimated and underestimated to the greatest extent in these two cases respectively. Thus, the estimation error of an arbitrary system must be no more than the difference between $MTTF_{\min}$ and $MTTF_{\max}$, namely,

$$\begin{aligned} \varepsilon_{MTTF} &\equiv |MTTF - MTTF_{\text{approx}}| \\ &\leq MTTF_{\max} - MTTF_{\min} \end{aligned} \tag{3.19}$$

Substituting Eq. (3.17) and Eq. (3.18) into this expression we have

$$\varepsilon_{MTTF} \leq \mathcal{R}(\Theta \cdot \Omega \cdot 0 \cdot \tau_d) \cdot \tau_d = \tau_d \tag{3.20}$$

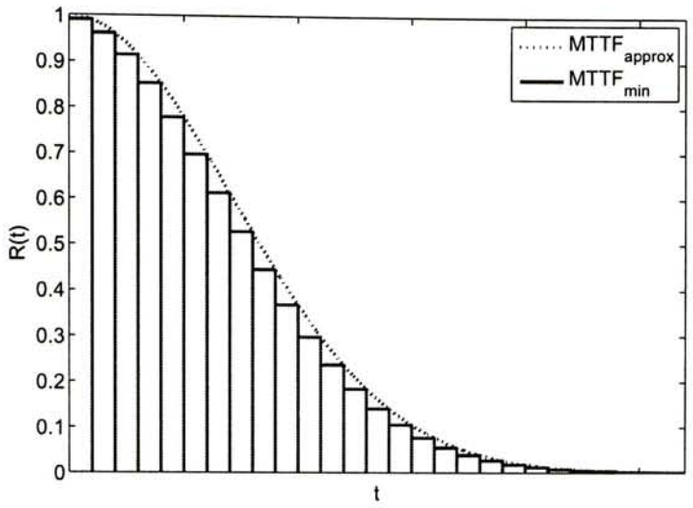
Hence the relative error has upper bound

$$\eta_{MTTF} \leq \frac{\varepsilon_{MTTF}}{MTTF} = \frac{\tau_d}{MTTF} \tag{3.21}$$

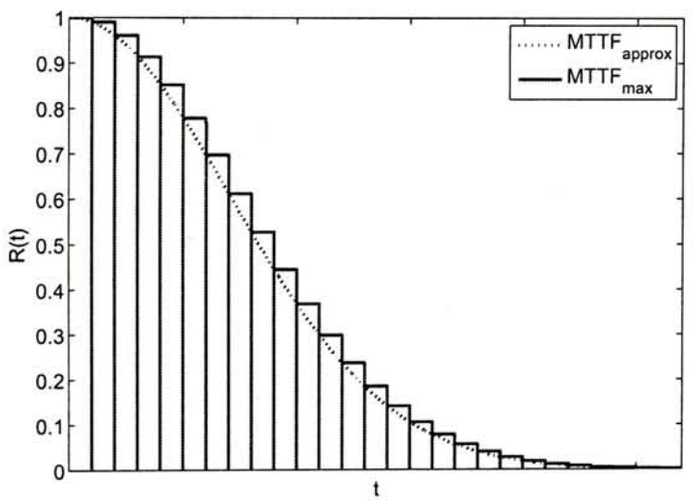
Since η_{MTTF} is proportional to τ_d, in case of $\tau_d \ll MTTF$ the estimation error can be negligible (or $\eta_{MTTF} \approx 0$). In other words, as long as the time duration of representative workload is much smaller than the expected service life (in the range of years), the approximation applied in *AgeSim* is perfectly acceptable.

3.4.4 Numerical Validation

In this section, we provide some numerical evidences to demonstrate the accuracy of the proposed method. Consider the aforementioned 9-core processor again. We compute its aging rate Ω with the trace file for one hour and then use Ω to calculate the system's lifetime reliability according to Eq. (3.13). For comparison, we have also tried to calculate the reliability according to the method used in [63]. Here, we trace the system operations for one hour and fill the system's service lifetime with workloads that are consistent to the system's usage strategy. We then use the average temperature value to calculate the lifetime reliability of the system. Both are compared with the results calculated by the definition of Weibull failure distribution directly, which has exactly the same workload as that for evaluating the method in [63]. The system lifetime reliability obtained using these three approaches are shown in Fig. 3.4. As can be observed from this figure, the proposed *AgeSim* can achieve almost identical reliability values when compared to that computed according to reliability definition. Using average temperature to obtain lifetime reliability, however, results in quite large errors.



(a) Overestimation



(b) Underestimation

Figure 3.3: Estimation Error in MTTF.

3.4.5 Miscellaneous

Besides the aforementioned mean time to failure, many interesting lifetime reliability metrics can be derived from the proposed simulation framework. This is because we are able to express the reliability at an arbitrary time t with the aging rate (see Eq. (3.12)), which enables the derivation of any reliability metrics, such

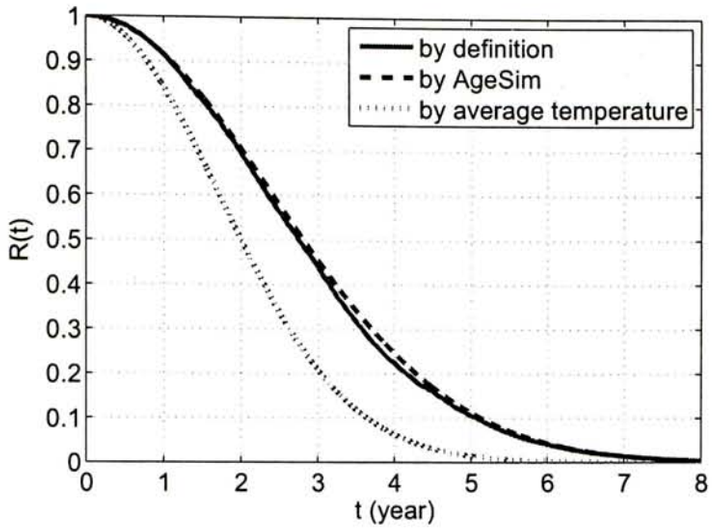


Figure 3.4: Accuracy Comparison.

as the reliability and failure rate at a certain time. For example, some design specifications set the lower bound of reliability at the end of warranty, i.e., $R(t_w) \leq R_{\text{req}}$. With *AgeSim* we can simply verify whether the following condition is met, where Ω is the proposed aging rate.

$$R(t_w) = \mathcal{R}\left(\Theta \cdot \Omega \cdot t_w\right) \leq R_{\text{req}} \tag{3.22}$$

If the failure rate at time t is an important metric in some situation, we can express it by substituting Eq. (3.12) into the following definition, namely, [106]

$$h(t) = -\frac{\mathrm{d}R(t)}{\mathrm{d}t} \cdot \frac{1}{R(t)} \tag{3.23}$$

It is also worth noting that the proposed method could be easily extended to analyze systems with multiple representative workloads (e.g., multi-mode MPSoCs [79]). We can organize these workloads into a hyper-workload according to their occurrence probabilities, and then take it as the input to *AgeSim*. Alternatively, we can extract the aging rate Ω_i and occurrence probability p_i for every execution

mode with representative workload i. Similar to the above mathematical deduction, the unified aging rate is simply

$$\Omega_u = \sum_i \Omega_i \cdot p_i \tag{3.24}$$

3.5 Lifetime Reliability Model for MPSoCs with Redundancy

In this section, we extend the previous model to analyze MPSoCs with redundant processor cores. Consider an MPSoC containing a set of $\{1, \cdots, n\}$ identical processor cores and it functions if no less than k components are good (e.g., Sony playstation game console requires seven out of eight synergistic processing elements in Cell processor to function [93]). The usage strategy of such a system can be changed a few times during its service life. For example, for gracefully degrading system, initially all cores are good ones and they share the system workload. Once a core fails, the workloads originally assigned to it need to be shared by the surviving cores, leading to heavier stress on them. Therefore, according to the usage strategy of cores, we divide the time horizon into several stages. In each stage, the usage of each core follows a fixed strategy.

Let us use $R_{i,\ell}(t)$ to denote the reliability of core i at stage ℓ. It depends on not only the characteristic at stage ℓ, but also that at previous stages. Without loss of generality, we assume core i can be in a series of states S_j at stage j. Accordingly, the aging rate and probability in state $s \in S_j$ are referred as $\Omega_{s,i,j}$ and $\pi_{s,i,j}$, respectively. With these notations, the reliability of core i at stage ℓ can be expressed as

$$R_{i,\ell}(t) = \mathcal{R}\Big(\Theta \cdot \big[\sum_{j=0}^{\ell-1} \sum_{s \in S_j} \pi_{s,i,j} \cdot \Omega_{s,i,j} \cdot (t_{j+1} - t_j) + \sum_{a \in S_\ell} \pi_{s,i,\ell} \cdot \Omega_{s,i,\ell} \cdot (t - t_\ell)\big]\Big) \tag{3.25}$$

It is important to note that probably not all cores are surviving or in power-on state at this stage. Let $\mathcal{L}_\ell$ be the set of power-on surviving cores at stage ℓ. If any one of them fails, the system will leave stage ℓ and enter stage $(\ell+1)$ or becomes faulty. Therefore, the conditional probability that the system remains in stage ℓ at time t provided the past events $\mathbf{h}_\ell$ is given by

$$P_\ell^{sys}(t|\mathbf{h}_\ell) = \prod_{i \in \mathcal{L}_\ell} R_{i,\ell}(t) \tag{3.26}$$

The history $\mathbf{h}_\ell$ can be characterized by two vectors: the events $\mathbf{e}_\ell = \{e_1, \cdots, e_\ell\}$ and their occurrence time $\mathbf{t}_\ell = \{t_1, \cdots, t_\ell\}$. For instance, $\mathbf{e}_2 = \{$core 15 fails, core 6 fails$\}$ and $\mathbf{t}_2 = \{t_1, t_2\}$ represent that at time t_1 core 15 fails and at time t_2 core 6 fails. The vector $\mathbf{e}_\ell$ directly affects the set of good cores with power supply $\mathcal{L}_\ell$. We can therefore rewrite Eq. (3.26) as

$$P_\ell^{sys}(t|\mathbf{t}_\ell; \mathbf{e}_\ell) = \prod_{i \in \mathcal{L}(\mathbf{e}_\ell)} R_{i,\ell}(t) \tag{3.27}$$

To uncondition it, we consider vectors $\mathbf{e}_\ell$ and $\mathbf{t}_\ell$ separately. We do not know the exact occurrence time of the past ℓ failures, but we are certain that the ℓ^{th} event must occur before current time t (i.e., within time $[0,t]$), the $(\ell-1)^{\text{st}}$ event occurs before the ℓ^{th} one (i.e., in $[0,t_\ell]$). Hence, we have an inequality that $0 < t_1 < t_2 < \cdots < t_{\ell-1} < t_\ell$. On the other hand, since all power-on cores are likely to have failures, the possible first ℓ events of the system may not be unique. To include all possible cases, we denote them by a set $\mathcal{E}_\ell$. By the theorem of total probability, the unconditioned probability is

$$P_{\ell}^{sys}(t) = \int_0^t \mathrm{d}t_{\ell} \int_0^{t_{\ell}} \mathrm{d}t_{\ell-1} \cdots \int_0^{t_2} \mathrm{d}t_1 \sum_{\mathbf{e}_{\ell} \in \mathcal{E}_{\ell}} P_{\ell}^{sys}(t|\mathbf{t}_{\ell}; \mathbf{e}_{\ell}) \Pr[\mathbf{t}_{\ell}; \mathbf{e}_{\ell}] \tag{3.28}$$

Since our redundant system functions if no less than k cores are good, the reliability of such a system is equivalent to the sum of probability for a series of events that exactly ℓ core failures happen before time t. Its service life hence can be calculated as

$$MTTF^{sys} = \int_0^{\infty} \sum_{\ell=0}^{n-k} P_{\ell}^{sys}(t) \mathrm{d}t \tag{3.29}$$

3.6 Case Studies

In this section, we conduct four case studies to show the flexibility and effectiveness of the proposed *AgeSim* simulation framework. Due to the lack of public benchmark workloads, we use high-level synthetic workloads in our simulation, which are an application flow consisting of a large number of applications. For each application, their power consumptions at each execution state are given. We evaluate the lifetime reliability of processor-based SoCs with various system load ρ and it is obtained as follows. The application arrival to the system is assumed as a Poisson process with arrival rate λ, while the service time is an exponential distribution with mean $1/\mu$ in our case studies[1]. Denoting by n the number of cores in the system, system load ρ is defined as $\lambda/n\mu$. In practice, designers should provide the above information when running representative workloads on their systems.

Our framework is applicable for any failure mechanism model or the combination of these models. Due to the lack of public data on the relative weights for various failure mechanisms, however, we select to use electromigration model presented in [39] in our case studies and the parameters are set as follows: the

[1]Our framework is applicable for any application flow characteristics.

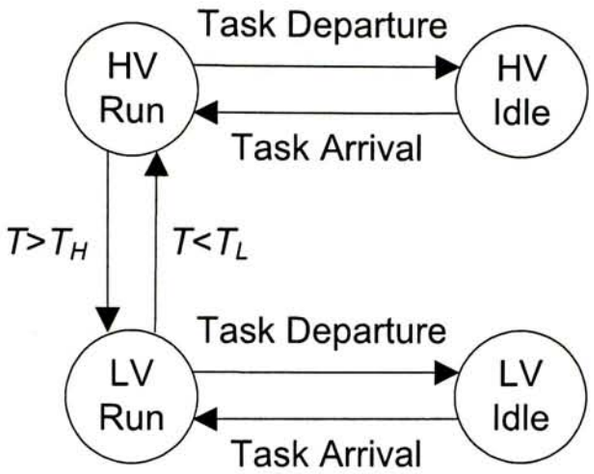
cross-sectional area of conductor $A_c = 6.4 \times 10^{-8} cm^2$, the current density $J = 1.5 \times 10^6 A/cm^2$ and the activation energy $E_a = 0.48eV$. In addition, we use Weibull distribution to describe wear-out effect with shape parameter $\beta = 4.0$, implying increasing failure rate with respect to time.

3.6.1 Dynamic Voltage and Frequency Scaling

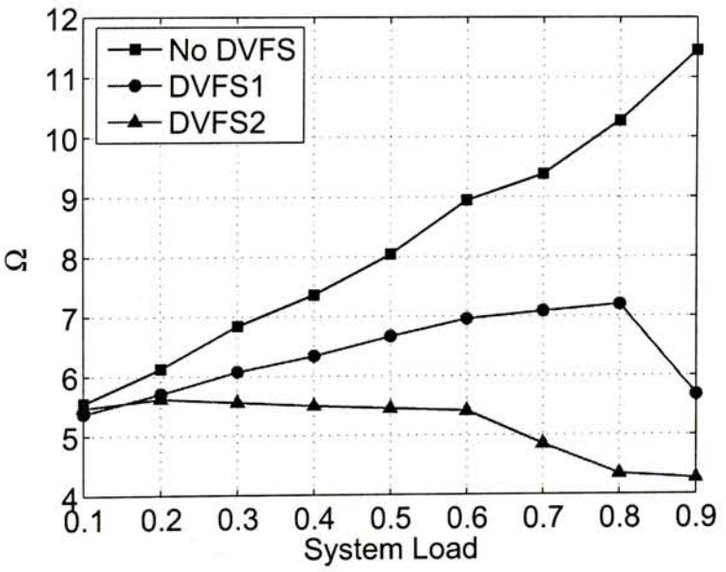
We first show the basic functionality of *AgeSim* with this case study, that is, it can be used to characterize the SoC with single processor core. In this experiment, three potential DPM policies are compared.

With DVFS, a processor core can be in one of four states: high voltage run, low voltage run, high voltage idle, and low voltage idle, as shown in Fig. 3.5(a). Here, high voltage suggests the supply voltage V_{dd}, while low voltage corresponds to $90\%V_{dd}$ or $80\%V_{dd}$ (denoted as DVFS1 and DVFS2, respectively). A core is in run state if it has applications to perform, while in idle state otherwise. When the processor's temperature is higher/lower than a threshold T_H/T_L, it decreases/increases its supply voltage and frequency. The processor's voltage does not change for the transitions between run and idle state due to the associated high overhead. We set $T_H = 348.15K$ and $T_L = 338.15K$. Based on the model proposed in [20], the time required for voltage changes in DVFS1 and DVFS2 is $22\mu s$ and $44\mu s$, respectively. The frequency and power consumption in various states are computed according to the model presented in [13].

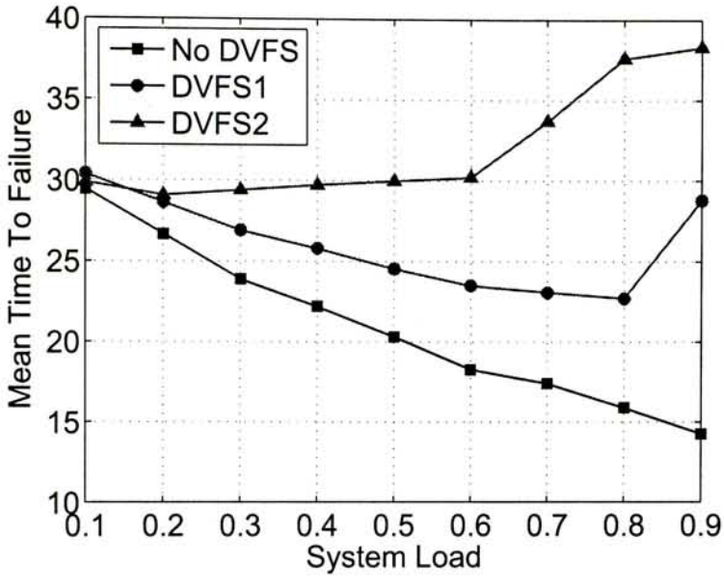
Three cases are compared: DVFS1, DVFS2, and no DVFS. Fig. 3.5(b) shows the lifetime reliability metrics in these three cases with different system workloads and their expected service lives are shown in Fig. 3.5(c). When the system load is only 0.1, the aging rates caused by three configurations are almost the same. This is because, when the workload is too light, the DVFS policy is applied only in very rare cases. In other words, the core alternates between high voltage run



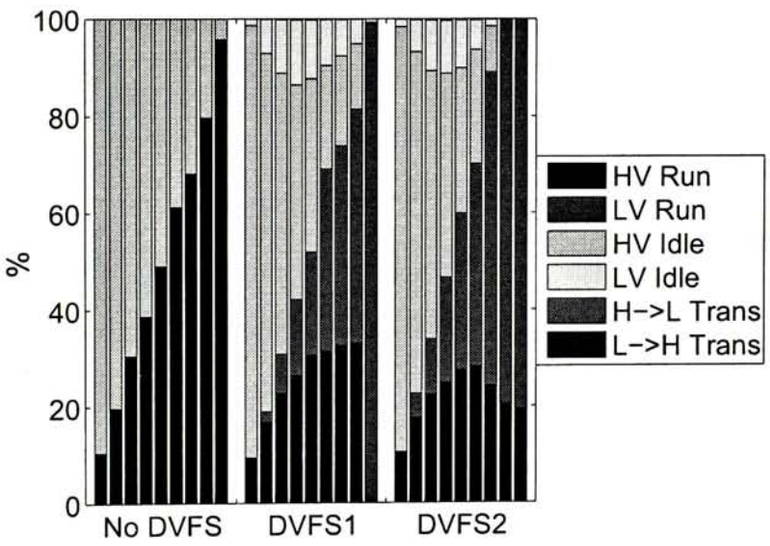
(a) Finite State Machine



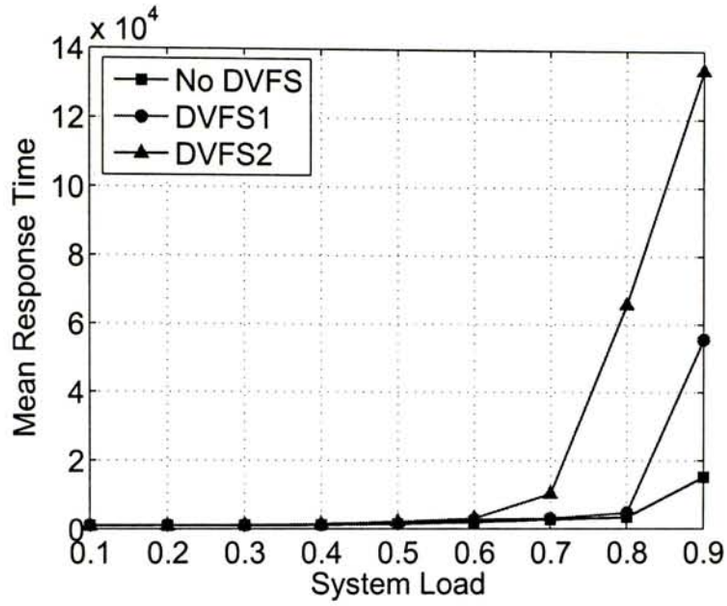
(b) Aging Rate



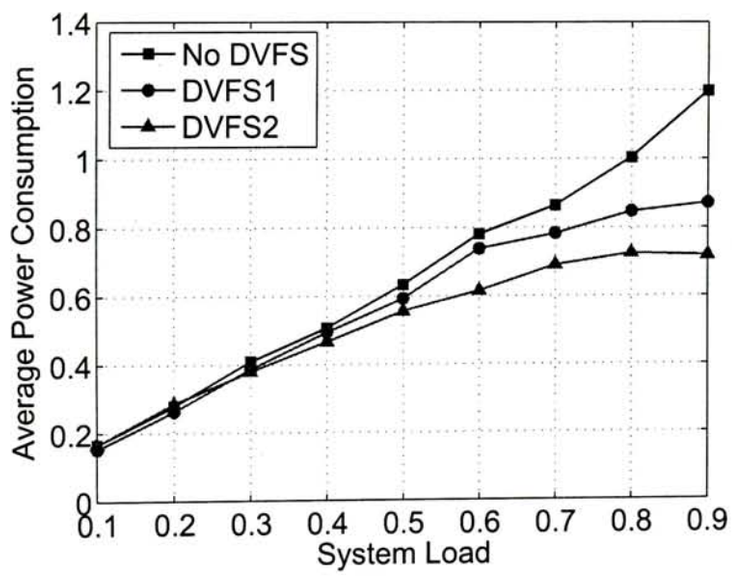
(c) MTTF



(d) Accumulated Time in Each State



(e) Performance



(f) Power Consumption

Figure 3.5: The Impact of Dynamic Voltage Frequency Scaling.

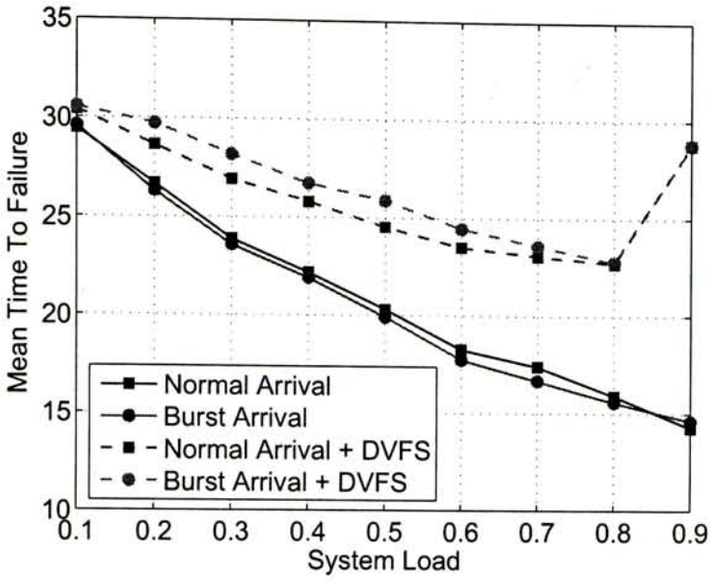
and high voltage idle states in the major portion of its lifetime, and seldom enters the low voltage states even if DVFS is used (see Fig. 3.5(d)). Note that, the bars in each group in Fig. 3.5(d) (e.g., nine bars on the left side for *No DVFS* case) represents system load 0.1-0.9 from left to right. With DVFS1, when the system load increases, as long as the workload is not too heavy, the aging rate increases, but the growth rate is lower than that without DVFS due to the fact that the processor spends more and more time in low voltage states with the increase of workloads. An interesting phenomenon is that when the system load increases to 0.9, the aging rate of DVFS1 case decreases. We attribute it to the fact that the processor remains in the low voltage states all the time with such high workload (see Fig. 3.5(d)). Next, we move to consider DVFS2, wherein frequency, voltage, and power consumption in low voltage states are much smaller than those with DVFS1. In this case, when the system load is greater than 0.2, its aging rate starts to decrease. This is due to the fact that the circuit spend more time in low voltage state with the increase of workloads, while the wear-out stress in low voltage run state becomes even lower than that in high voltage idle state with DVFS2.

As discussed in Section 3.3, the performance and power consumption can also be obtained with *AgeSim*. The results for all cases are shown in Fig. 3.5(e) and Fig. 3.5(f), respectively. We observe severe performance degradation and some power savings by applying DVFS policy when the system load is high (e.g., $\rho \geq 0.8$).

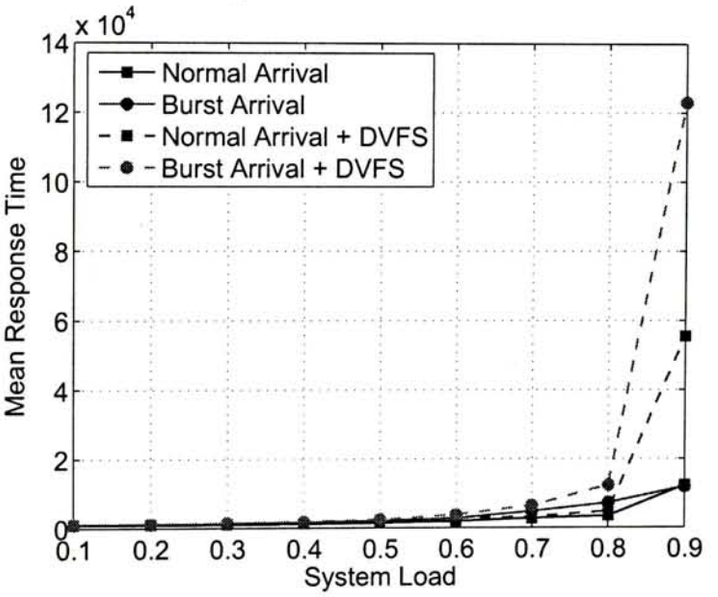
3.6.2 Burst Task Arrival

When a system is used to process the application flows with different characteristics, its lifetime reliability might be different, which can also be captured by *AgeSim*.

In this case study, we consider the normal task arrival that has been introduced



(a) Mean Time to Failure



(b) Performance

Figure 3.6: The Impact of Burst Task Arrival.

in the experimental setup and the burst task arrival with which the application arrival to the system follows Poisson process and every time X tasks arrive simultaneously. We set $\Pr\{X = 1\} = 0.3$, $\Pr\{X = 2\} = 0.5$, and $\Pr\{X = 3\} = 0.2$. The system load ρ in the "burst" cases is defined as $\mathrm{E}[X]\lambda/n\mu$. As shown in Fig. 3.6(a),

given the same system load the dynamic voltage and frequency scaling policy results in more significant lifetime extension in case of burst task arrival, especially when the system workload is not heavy. We attribute this phenomenon to the fact that the system tends to frequently alternate between run and idle states in case of normal arrival while less frequently in the other case. Thus with burst task arrival the system has sufficient time to cooling down and hence the temperature in the idle states is lower. When the system workload increases to 0.8, the system seldom remains in the idle states with either burst or normal arrival and therefore the lifetime reliability enhancement in two cases are almost the same. *AgeSim* also show that, as expected, the performance of the system with burst arrival is much worse than that with normal arrival in case of heavy system workload, as plotted in Fig. 3.6(b). This observation is also obtain from the trace file for lifetime reliability evaluation.

3.6.3 Task Allocation on Multi-Core Processors

AgeSim is also applicable for multi-core system and it can evaluate the influence of system-level policies, such as task allocation strategies.

Due to process variation, processor cores in a homogeneous multi-core processor may have different operational frequencies. When an application arrives at the processor, a simple method is to randomly choose any available core to process this task, namely *random strategy*. To optimize performance, however, one may use the available core with the highest frequency to process it. We call this strategy as *performance-aware strategy*. Two application schedulers are implemented in our simulator accordingly.

Considering a 9-core processor with cores running at different frequencies (the maximum difference is 30%), we analyze the aging rate of each core with *AgeSim* and show the result in Fig. 3.7(a). When performance-aware strategy is used, we

can observe significant variance among aging rates of different cores, especially when system workload is low. This is expected because those high-frequency cores are used much more often than those low-frequency ones under such circumstances. In contrast, the difference of cores' aging rates is relatively small if random allocation is assumed. Their impact on MTTF can be seen in Fig. 3.7(b), in which the lifetime reduction caused by unbalanced usage is quite serious when the system load is smaller than 0.5. When the system workloads become very high (e.g., $\rho = 0.9$), the aging rates with the two different allocation strategies are similar. This is because all processor cores are busy in most of their lifetime no matter which allocation strategy is chosen. In addition, the benefit of the performance-aware allocation strategy in terms of performance is shown in Fig. 3.7(c), which has a shorter mean response time.

3.6.4 Timeout Policy on Multi-Core Processors with Gracefully Degrading Redundancy

We have extended the proposed model to analyze the system with redundancy in Section 3.5. The case study presented in this section is to demonstrate a concrete instance.

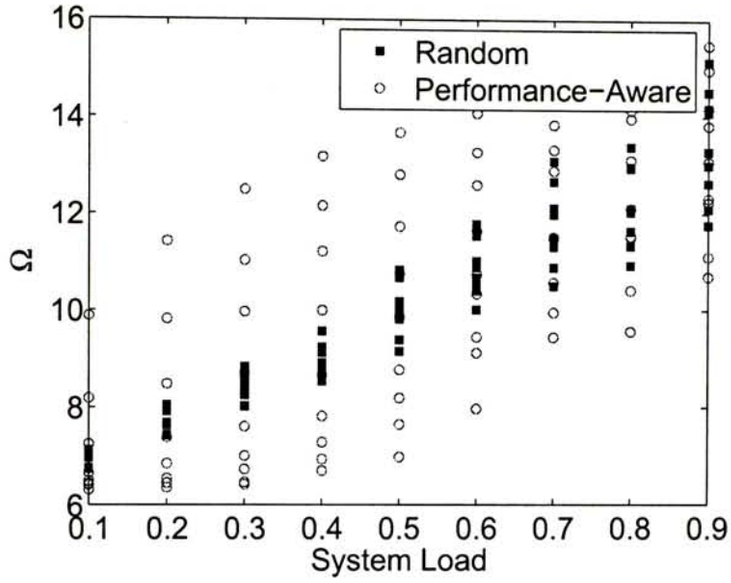
Timeout policy is widely-used for power savings in electronic products. This experiment examines its impact on lifetime reliability. We consider a 8-out-of-9: G system, meaning that 9 cores are fabricated in such a system, and as long as no less than 8 cores are good the entire system is functioning. The system is used in a gracefully degrading manner. That is, all good cores share the workloads. When a core is detected as defective, the system is reconfigured as an 8-core processor. We analyze the aging rate of a single processor core for the two cases when 9 cores and 8 cores share workloads, and then compute the expected service life of the system using the proposed model in Section 3.4.

Fig. 3.8 shows our simulation results. We observe significant lifetime extension with timeout policy when the system workload is low (e.g., lower than 0.4). This is because cores spend a large share of its lifetime in *power down* state (see Fig. 3.8(b)) with negligible failure rates. With the increase of workloads, the timeout policy cannot provide much benefit in terms of lifetime reliability. This is because, as shown in Fig. 3.8(b), when processor cores become busier, they seldom enter the *power-down* state.

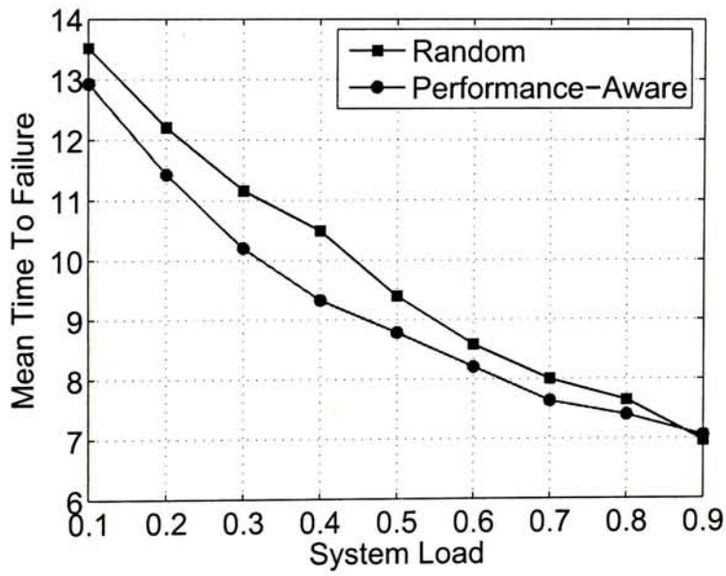
3.7 Conclusion

With the relentless scaling of CMOS technology, the lifetime reliability of processor-based SoCs has become a serious concern for the industry. To meet the reliability requirement, designers need to know the impact of various usage strategies on the system lifetime. To facilitate this process, this paper proposes an accurate yet efficient simulation framework, which is applicable for evaluating any DPM/DTM policies, application flow characteristics and even task allocation algorithms, by tracing the system's reliability-related factors for representative workloads only. Four case studies are conducted to demonstrate the effectiveness of the proposed framework.

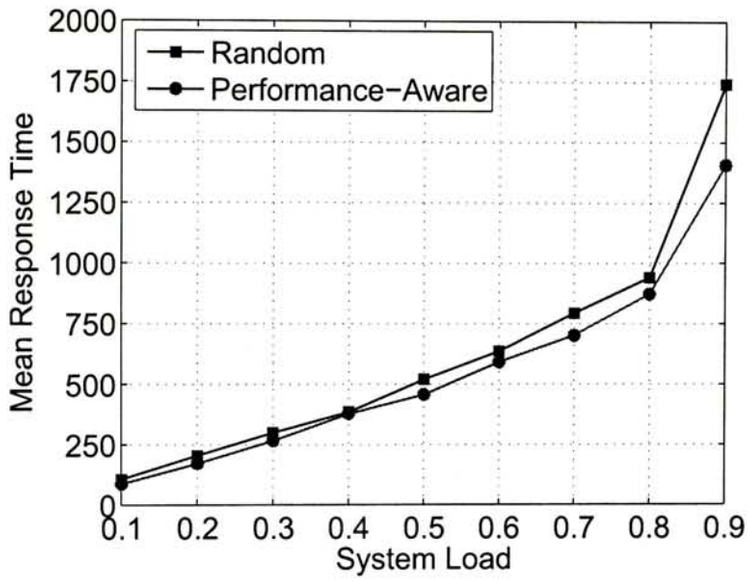
□ End of chapter.



(a) Aging Rate of Cores

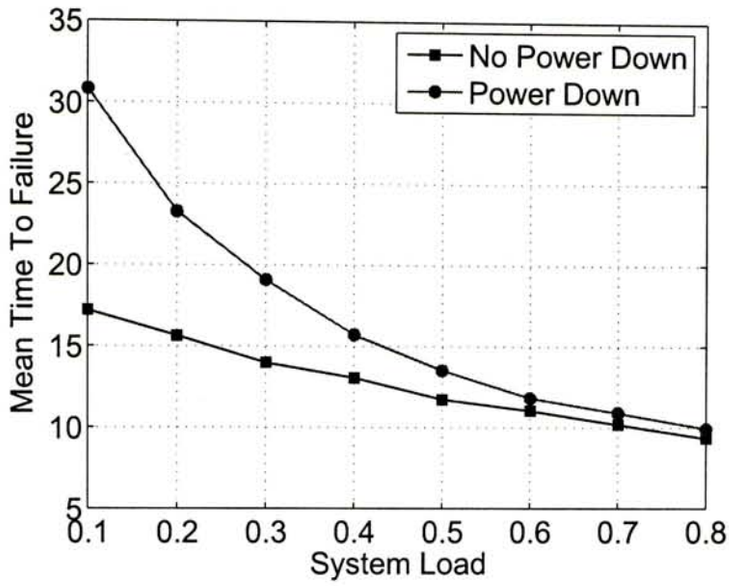


(b) Mean Time to Failure

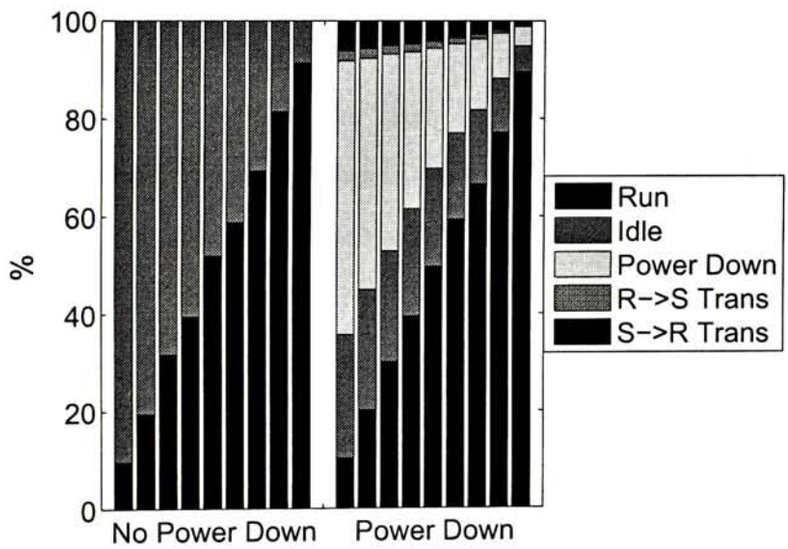


(c) Performance

Figure 3.7: Comparison of Task Allocation Schemes.



(a) Mean Time to Failure



(b) Accumulated Time in Each State

Figure 3.8: The Impact of Timeout Policy.

Chapter 4

Evaluating Redundancy Schemes

The content in this chapter is included in the proceedings of *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* 2010 [48].

4.1 Introduction

As technology advances, industry has started to employ multiple processor cores on a single silicon die in order to improve performance through parallel execution. Such chip multiprocessors, also known as multi-core or many-core processors (depending on the number of cores on the die), being much more power-efficient than unicore processors with extremely high frequency, have become increasingly popular [36]. Several large-scale CMPs, such as the nVidia 128-core GeForce 8800 GPU [77] and the Intel 80-core teraflop processor [108], have been built in recent years. Various research groups have also predicted that thousand-core processors will be commercialized over the next decade [4, 17].

While the relentless technology scaling has brought with it enhanced functionality and improved performance in every new generation, the wear-out-related errors have an increasingly adverse effect to result in permanent failures of the cir-

cuits. The lifetime reliability of today's high-performance integrated circuits has thus become a serious concern for the industry [95, 104, 90, 62].

One method to obtain defect-tolerance capabilities is to incorporate redundant circuits in a system and use them as replacements when some units are faulty. This strategy has been proved to be very effective to keep a high manufacturing yield for large-scale ICs. In particular, for multi-core processors, as a single processor core becomes inexpensive when compared to the entire system, employing core-level redundancy is a more attractive solution than introducing complex microarchitecture-level redundancy and has been practiced in the industry. For example, the 192-core Cisco Metro network processor [25] contains four redundant cores for yield improvement. Redundancy can be also used for lifetime reliability enhancement of IC products. Again, while introducing microarchitecture-level redundant structures (e.g., [97]) is cost-effective for multi-core processors with a few embedded cores, for large-scale CMPs containing tens to hundreds of processor cores, core-level redundancy is an attractive solution to extend their lifetimes.

For multi-core processors with core-level redundancy, there are many ways to make use of the redundant cores. We can configure some cores as standbys and use them only when some of the active cores fail. We can also activate all the available cores from the beginning and remove the faulty cores during the system's lifetime. Moreover, we have the freedom to dynamically configure which cores to serve as active cores and which cores to serve as spares at a specific time. As ICs' wear-out-related failure rates are significantly related to their operational conditions such as temperature and/or voltage, the above strategies result in different stress on the aging of the processors. How to characterize the lifetime reliability of multi-core processors with different usages is therefore an important and relevant problem.

To address the above problem, in this paper, we explicitly consider the temperature variations caused by workloads in our analytical model to estimate the

lifetime reliability of multi-core processors with various redundancy schemes. To be specific, we introduce a parameter namely *wear-out rate* to reflect a core's aging effect in its different operational state, which is computed with the temperature distribution of the core. We then model the lifetime reliability of multi-core processors using wear-out rates. Finally, extensive experiments are conducted to compare multi-core processors in terms of both lifetime reliability and performance, under various workloads, service time distributions, and redundancy configurations.

The remainder of this paper is organized as follows. Section 4.2 reviews related prior work and motivates this paper. Our analytical model for the lifetime reliability of a processor core is then detailed in Section 4.3 and we use this model to investigate the impact of various redundancy schemes on the service life of multi-core processors in Section 4.4. Next, Section 4.5 and Section 4.6 present our experimental methodology and experimental results. Finally, Section 4.7 concludes this work.

4.2 Preliminaries and Motivation

4.2.1 Failure Mechanisms

Due to device wear-out, IC products suffer from various types of intrinsic failures, which manifest themselves after some time of operation and hence determine the circuits' service life. Major intrinsic failures include TDDB in the gate oxides, EM in the interconnects, NBTI stresses that shift PMOS transistor threshold voltages, and thermal cycling.

Many widely accepted reliability models for the above failure mechanisms at device- and circuit-level have been proposed and empirically validated by academia and industry [15, 35, 1, 2, 114] and it is shown that they were strongly related to the temperature and voltage applied to the circuit. These models, however, are

not readily applicable in characterizing the lifetime reliability of multi-core processors, because they assume constant temperature and voltage while these values vary significantly at runtime.

Failure rate function gives the conditional probability that a failure will happen for the first time at time t. It provides an instantaneous rate of failure and uniquely determines a reliability function, which gives the probability that a system will not fail up to time t. *Mean time to failure* is defined as the expected value of failure distribution and expressed in time (e.g., hours) per failure. In a special case when we have exponential failure distribution, MTTF is simply the inverse of the failure rate. The assumption for exponential failure distribution is widely-used in the literature mainly due to its easy computational tractability. In practice, we expect increasing failure rates as systems grow older and it is suggested to use non-exponential distributions, such as Weibull distribution and/or lognormal distribution, to describe the influence of hard errors [97, 1, 110].

4.2.2 Related Work and Motivation

While the fundamental causes of the above failure mechanisms have been studied for decades, it has recently re-attracted lots of research interests, due to their increasingly adverse effect with technology scaling.

Processor lifetime reliability is significantly affected by its operating conditions, which vary with different applications running on the processor. In [95, 97], Srinivasan *et al.* proposed an application-aware architecture-level model, namely *RAMP* model, which is able to dynamically track lifetime reliability of a processor according to changes in application behavior. In this model, the authors assumed a uniform device density over the chip and an identical vulnerability of devices to failure mechanisms. Later, Shin *et al.* [91] defined reference circuits and introduced a structure-aware model that takes the vulnerability of basic structures of

the microarchitecture (e.g., register files, latches and logic) to different types of failure mechanisms into account. It should be emphasized that both Srinivasan's and Shin's models target unicore architecture. Coskun *et al.* [29] introduced two analytical frameworks for the lifetime reliability of multi-core systems: a cycle-accurate simulation methodology and a statistical one, assuming uniform device density.

Some of the above models (e.g., [95, 29]) assumed exponential failure distributions (i.e., constant failure rate) and thus cannot capture the processors' accumulated aging effect. Consider NBTI as an example, the increase of threshold voltage ΔV_{th} at time t highly depends on the usage history of the transistors. The above models thus cannot predict the lifetime of the circuits accurately. In [97], the authors modeled processors with microarchitecture-level redundancy as series-parallel failure systems with lognormal failure distribution and used a simple MIN-MAX analysis to determine the system lifetime. This model, however, is not applicable for analyzing the lifetime reliability of multi-core processors with core-level redundancy. Firstly, it cannot reflect the load-sharing feature of multi-core processors. As highlighted in [72], a failure on the load-sharing system typically results in heavier workload on surviving components. The Monte Carlo simulation used in [97] cannot capture this feature. More importantly, series-parallel model is not applicable for many often-used configurations, such as standby redundant system, wherein some cores start their service life only when permanent core failures occur in the system.

Recently, Huang and Xu [46, 50] developed a high-level analytical model for the lifetime reliability of multi-core processors, which takes arbitrary failure distribution and load-sharing feature into account. In this work, a processor core is assumed to be in three possible states: *processing state*, *wait state*, and *spare state*, each corresponding to a unique failure distribution. The above assumption,

however, oversimplifies this problem because the lifetime reliability of a processor core highly depends on its operational temperature, which varies with different applications running on it. That is, even if two cores are in the same states and have the same usage history, they do not necessarily have the same failure rates. From this aspect, we need to extend the discrete states into a series of continuous states for more accurate estimation of the system's lifetime reliability, by taking the temperature and structural information that affect the system's lifetime reliability into account.

The above observations have motivated the work studied in this paper.

4.3 Proposed Analytical Model for the Lifetime Reliability of Processor Cores

As discussed earlier, the circuit wear-out effect are strongly related to its operational status such as temperature, voltage and frequency, which are not explicitly considered in the analytical model in [46, 50]. In this section, we first examine the impact of these factors on processor cores' lifetime reliability. Next, we consider the impact of workloads by mapping them into the different temperature distributions of the processor cores.

4.3.1 Impact of Temperature, Voltage, and Frequency

To examine the impact of temperature, supply voltage, and clock frequency on the wear-out effect of a single processor core, we start with the simplest case: no failures occur in the system up to time t. Under such circumstances, the task interarrival time distribution of a core is fixed up to time t.

We use the notation $\mathcal{R}(t,\theta)$ to denote a general reliability function, where θ represents the general scale parameter by which time t is divided and depends on

temperature and processor's execution mode. For example, the commonly-used Weibull failure distribution has the form $\mathcal{R}_{Weibull}(t,\theta) = e^{-(\frac{t}{\theta})^{\beta}}$. Without ambiguity, we hereafter drop the notation θ because of its generality and refer to the general reliability function as $\mathcal{R}(t)$. Note that, $\mathcal{R}(t)$ does not necessarily to be an exponential distribution.

Without loss of generality, we consider a core can be in any state s of set $\mathcal{S}$. An example of set $\mathcal{S}$ is defined in [46], namely, $\{processing, wait, spare\}$. Depending on a core's state, temperature, voltage, and frequency, let a subdivision of $[0,t]$ be a finite sequence $0 = \widetilde{t}_0 < \widetilde{t}_1 < \cdots < \widetilde{t}_d = t$, which partitions the interval $[0,t]$ into d sub-intervals. In each time sub-interval $[\widetilde{t}_j, \widetilde{t}_{j+1}]$ (referred as sub-interval j hereafter), the core remains in the same state, and its voltage and frequency keep unchanged. We denote the state, voltage, and frequency in sub-interval j as s_j, V_j and f_j, respectively. In addition, the temperature variation in every sub-interval is very small. Formally, since temperature is function of time, we use $T(t)$ represent a core's temperature at time t. For all $\varepsilon > 0$ there exists $\delta > 0$ such that, if the largest partition $\max_j(\widetilde{t}_{j+1} - \widetilde{t}_j) < \delta$ then for all j the difference between $\max_{\widetilde{t}_j < t < \widetilde{t}_{j+1}} T(t)$ and $\min_{\widetilde{t}_j < t < \widetilde{t}_{j+1}} T(t)$ is less than ε. Denote by $\Delta_j\widetilde{t}$ the difference $\widetilde{t}_{j+1} - \widetilde{t}_j$ and by T_j^* any value of T such that $\min_{\widetilde{t}_j < t < \widetilde{t}_{j+1}} T(t) \leq T_j^* \leq \max_{\widetilde{t}_j < t < \widetilde{t}_{j+1}} T(t)$. The corresponding scale parameter is expressed as $\theta_{s_j}(T_j^*, V_j, f_j)$.

By the definition of reliability function, we express $R(t)$ in $[\widetilde{t}_0, \widetilde{t}_1]$ as

$$R(\tau) = \mathcal{R}\left(\frac{\theta}{\theta_{s_0}(T_0^*, V_0, f_0)} \cdot (\tau - \widetilde{t}_0)\right), \qquad \widetilde{t}_0 \leq \tau \leq \widetilde{t}_1 \tag{4.1}$$

Next, substituting $\tau = \widetilde{t}_1$ into Eq. (4.1) yields the core's reliability at the end of the first sub-interval, i.e.,

$$R(\widetilde{t}_1) = \mathcal{R}\left(\frac{\theta}{\theta_{s_0}(T_0^*, V_0, f_0)} \cdot (\widetilde{t}_1 - \widetilde{t}_0)\right) \tag{4.2}$$

Because of the continuity of reliability function, the reliability at the beginning

of the second sub-interval has the same value as Eq. (4.2). With this condition, we express the reliability in the second sub-interval $[\widetilde{t}_1, \widetilde{t}_2]$ as

$$R(\tau) = \mathcal{R}\left(\frac{\theta}{\theta_{s_1}(T_1^*, V_1, f_1)} \cdot (\tau - \widetilde{t}_1) + \frac{\theta}{\theta_{s_0}(T_0^*, V_0, f_0)} \cdot (\widetilde{t}_1 - \widetilde{t}_0)\right),$$

$$\widetilde{t}_1 \leq \tau \leq \widetilde{t}_2 \tag{4.3}$$

By the same argument, at time t the reliability is given by

$$\begin{aligned} R(t) &= \mathcal{R}\left(\theta \cdot \sum_{j=0}^{d-1} \frac{1}{\theta_{s_j}(T_j^*, V_j, f_j)} \cdot (\widetilde{t}_{j+1} - \widetilde{t}_j)\right) \\ &= \mathcal{R}\left(\theta \cdot \sum_{j=0}^{d-1} \frac{1}{\theta_{s_j}(T_j^*, V_j, f_j)} \cdot \Delta_j \widetilde{t}\right) \end{aligned} \tag{4.4}$$

By the limiting process which is illustrated as follow, we have

$$R(t) = \mathcal{R}\left(\theta \cdot \lim_{\substack{d \to \infty \\ \max \Delta_j t \to 0}} \sum_{j=0}^{d-1} \frac{1}{\theta_{s_j}(T_j^*, V_j, f_j)} \cdot \Delta_j \widetilde{t}\right) \tag{4.5}$$

In this expression, the state parameter s_j is in the set $\mathcal{S}$ for any j. To simplify Eq. (4.5), we introduce a filter function over time horizon $\mathbb{1}_s(\widetilde{t}, V, f)$ such that it equals 1 if the core is in state s with voltage V and frequency f at time $\widetilde{t}$ while 0 otherwise. With this notation, Eq. (4.5) comes down to

$$R(t) = \mathcal{R}\left(\theta \cdot \sum_{s \in \mathcal{S}} \sum_{V} \sum_{f} \int_0^t \frac{\mathbb{1}_s(\widetilde{t}, V, f)}{\theta_s(T, V, f)} \mathrm{d}\widetilde{t}\right) \tag{4.6}$$

In this equation, we integrate $\frac{1}{\theta_s(T,V,f)}$ over $\widetilde{t}$. To integrate over $\mathrm{d}T$ (T is a function of $\widetilde{t}$), we denote by $\psi_s(T, V, f)\mathrm{d}T$ the accumulated time in state s with voltage V and frequency f in an infinitesimal temperature interval $\mathrm{d}T$ around T. We use it to substitute $\mathrm{d}\widetilde{t}$ and change lower and upper limits of integration accordingly, yielding

$$R(t) = \mathcal{R}\left(\theta \cdot \sum_{s\in S}\sum_{V}\sum_{f}\int_{0}^{\infty} \frac{1}{\theta_s(T,V,f)} \cdot \psi_s(T,V,f) \cdot \mathrm{d}T\right) \tag{4.7}$$

Further, we use $\nu_s(T,V,f)$ to represent the probability density function (p.d.f.) of a core with temperature T, given the core is in state s. Also, π_s is defined as the probability a core being in state s. Thus, the fraction of accumulated time within which the core falls in a infinitesimal interval $\mathrm{d}T$ at T and is in state s can be approximated by $\pi_s \cdot \nu_s(T,V,f) \cdot \mathrm{d}T$. Hence, Eq. (4.7) can be rewritten as

$$R(t) = \mathcal{R}\left(\theta \cdot \sum_{s\in S}\sum_{V}\sum_{f}\int_{0}^{\infty} \frac{1}{\theta_s(T,V,f)} \cdot \pi_s \cdot \nu_s(T,V,f) \cdot t \cdot \mathrm{d}T\right) \tag{4.8}$$

Because π_s and t are independent of V, f and T, they can be moved outside of the corresponding integral and summation signs to obtain

$$R(t) = \mathcal{R}\left(\theta \cdot \sum_{s\in S} \pi_s \cdot \left(\sum_{V}\sum_{f}\int_{0}^{\infty} \frac{1}{\theta_s(T,V,f)} \cdot \nu_s(T,V,f)\mathrm{d}T\right) \cdot t\right) \tag{4.9}$$

Now, we are ready to introduce the formal definition of *wear-out rate* in state s, a quantity that describes the rate of core suffering from wear-out effect, namely,

$$\Omega_s \equiv \sum_{V}\sum_{f}\int_{0}^{\infty} \frac{\nu_s(T,V,f)}{\theta_s(T,V,f)}\mathrm{d}T \tag{4.10}$$

Using Ω_s, Eq. (4.9) can be rewritten as

$$R(t) = \mathcal{R}\left(\theta \cdot \sum_{s\in S} \pi_s \cdot \Omega_s \cdot t\right) \tag{4.11}$$

Clearly, ν_s follows a constraint that

$$\sum_{V}\sum_{f}\int_{0}^{\infty} \nu_s(T,V,f)\mathrm{d}T = 1 \tag{4.12}$$

Here, $\nu_s(T,V,f)$ is the conditional p.d.f. of temperature T, voltage V, and frequency f with given state s. According to the theorem of total probability, it is possible for us to drop s from notation Ω_s and express wear-out rate in a concise form. As both θ and t are independent of wear-out rate, from Eq. (4.11) we define

$$\Omega = \sum_{s \in S} \pi_s \cdot \Omega_s = \int_0^\infty \sum_{s \in S} \sum_V \sum_f \frac{\pi_s \cdot \nu_s(T,V,f)}{\theta_s(T,V,f)} \mathrm{d}T \tag{4.13}$$

Thus, Eq. (4.11) can be written as

$$R(t) = \mathcal{R}\left(\theta \cdot \Omega \cdot t\right) \tag{4.14}$$

In particular, if the core has the same frequency and voltage in various states other than in the cold standby mode, we can redefine scale parameter $\theta(T)$ according to these parameters. Since a core in cold standby state is switched off, its lifetime is close to infinity, i.e., $\theta_{spare}(T) \to \infty$. In other words, the wear-out rate contributed by this state is approximated to zero. Therefore, we are only interested in the temperature distribution given the core is not in cold standby, denoted as $\nu(t)$. For a core which is not set into cold standby state within a time interval, from Eq. (4.13), we have

$$\Omega = \int_0^\infty \frac{\nu(T)}{\theta(T)} \mathrm{d}T \tag{4.15}$$

where, temperature distribution $\nu(T)$ follows

$$\int_0^\infty \nu(T) \mathrm{d}T = 1 \tag{4.16}$$

4.3.2 Impact of Workloads

In many systems, the workload distribution of a core is not fixed. For instance, in a gracefully degrading multi-core processor with redundant cores, all cores share the

workload initially. Once a core fails, the system will be reconfigured in a degrading manner, which implies that every surviving core has greater workload or follows another interarrival time distribution. We therefore examine how workloads affect the wear-out rate in this section.

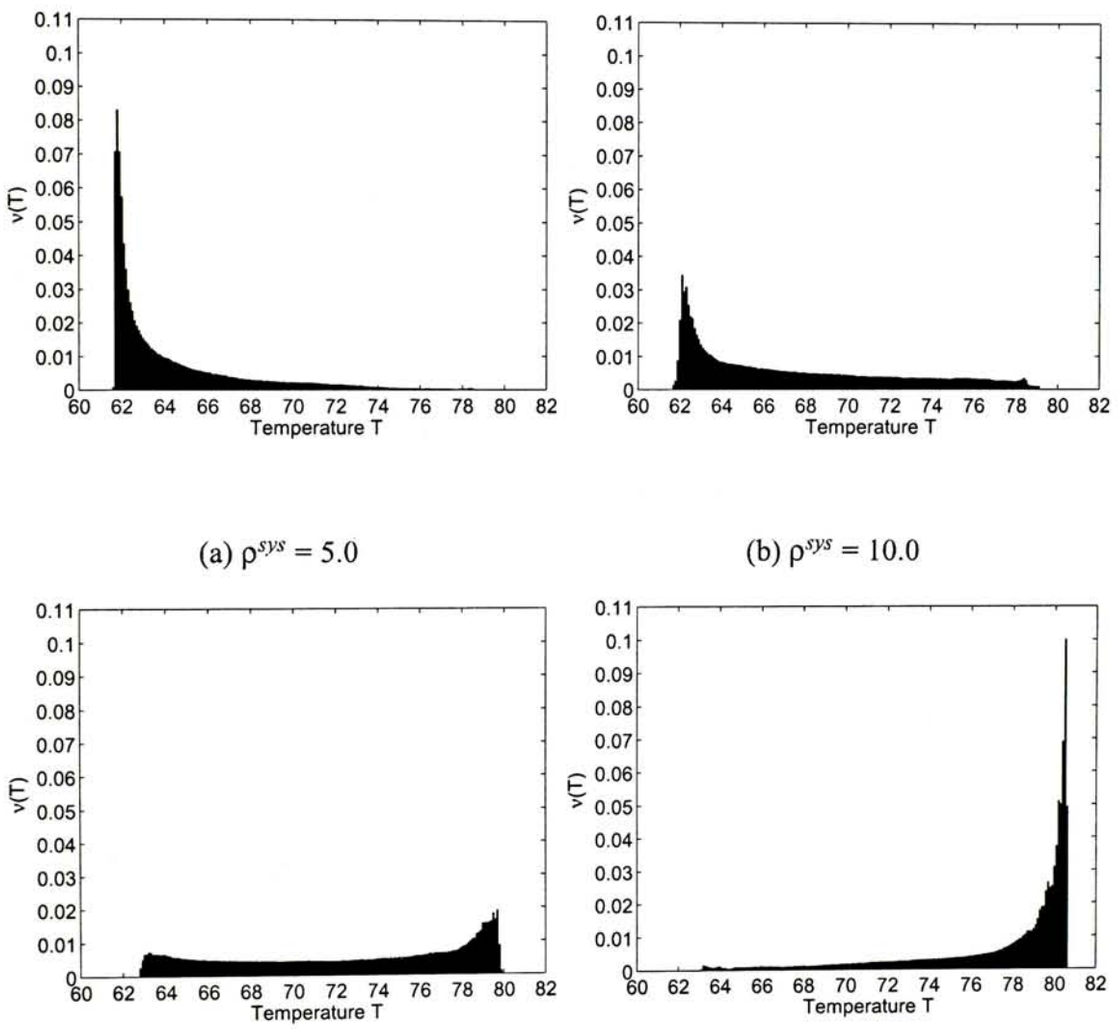
Remind the mathematical derivation of unified reliability function is independent of workload distribution. Suppose a set $\mathcal{M} = \{1, 2, \cdots, m\}$ of cores equally share the workload, the probability for core i ($1 \leq i \leq m$) to process any task is $\frac{1}{m}$. Thus, given the workload distribution of the entire system, it is easy to know every core's workload. In our model, it is reflected in temperature distributions $\nu_s(T, V, f)$ and hence the wear-out rate Ω. Fig. 4.1 shows typical temperature distributions of a core under various workloads ρ^{sys} (the formal definition is introduced in Section 4.5.1). The data is collected from HotSpot [92] for an application flow composed of 10,000 tasks. Without loss of generality, we assume every state s corresponds to a single supply voltage value V and clock frequency value f in this numerical experiment. We add a subscript m to indicate the quantity of cores which process workloads in the system. For example, Ω_{36} can be used to represent the wear-out rate of system that contains 36 active units.

We then present how to extend the definition of Ω, which is drawn from the expression of $\mathcal{R}(t)$ assuming the distribution is fixed from time zero up to t, to compute wear-out rate Ω_m for any active core quantity m. Under the same usage strategy, for the same system the difference in wear-out rate caused by different workload is reflected in temperature distributions $\nu_{s,m}(T, V, f)$ over T and the probabilities of a core being in various states $\pi_{s,m}$. Consequently, from Eq. (4.13) we have

$$\Omega_m = \int_0^{\infty} \sum_{s \in S} \sum_{V} \sum_{f} \frac{\pi_{s,m} \cdot \nu_{s,m}(T, V, f)}{\theta_s(T, V, f)} \mathrm{d}T \tag{4.17}$$

Even if a core has experienced other workload distribution before the current

one, Eq. (4.17) is able to capture the aging effect in this time interval. To take an example, suppose all n cores in a system equally share workload at the beginning, then one of them fails at time t_1 resulting in a heavier load on every surviving core. In this case, we can use Ω_n and Ω_{n-1} to represent the wear-out rate in two states respectively. From Eq. (4.14), the reliability of a surviving core at time $\hat{t}$



(a) $\rho^{sys} = 5.0$ (b) $\rho^{sys} = 10.0$

(c) $\rho^{sys} = 20.0$ (d) $\rho^{sys} = 30.0$

Figure 4.1: Temperature Distribution under Various Workloads (Exponential Service Time).

($\hat{t} \leq t_1$) is $R(\hat{t}) = \mathcal{R}(\theta \cdot \Omega_n \cdot \hat{t})$. Then, we analyze the second state. Since this core enters its second state at time t_1, its initial reliability of the second state is $R(t_1) = \mathcal{R}(\theta \cdot \Omega_n \cdot t_1) = \mathcal{R}(A_1)$, where $A_1 \equiv \theta \cdot \Omega_n \cdot t_1$. Sequentially, by the similar argument with that in Section 4.3.1, we can express the reliability of a surviving core at time t ($t > t_1$) as $R(t) = \mathcal{R}\left(\theta \cdot \Omega_{n-1} \cdot (t - t_1) + A_1\right)$.

4.4 Lifetime Reliability Analysis for Multi-core Processors with Various Redundancy Schemes

Modeling the lifetime reliability of multi-core processors with redundancy is more complicated. This is because, the status, workload and the corresponding failure rate of each core in a system can be time-varying, depending on the redundancy configuration and wear-out-related failure occurrences. Needless to say, to analyze various redundant multi-core systems, it is necessary to capture the above features. In this section,we focus on three redundant schemes and discuss their lifetime reliability models in detail. Then, we present how to extend the proposed model for heterogeneous multi-core systems.

4.4.1 Gracefully Degrading System (GDS)

In GDS, initially all n cores are configured as active units. When a core fails, the system will be reconfigured in a gracefully degrading manner, that is, the remaining $(n-1)$ good cores share the system workload. This process continues until there are only k good cores left. In that situation, if one more core fails, the entire system will be considered as faulty. The number of cores sharing workloads can be n, $n-1$, $\cdots$, k, and corresponding wear-out rates are Ω_n, Ω_{n-1}, $\cdots$, Ω_k, respectively. By extending deduction procedure presented in Section 4.3, for any surviving core at time t, given that the system contains $(n-\ell)$ good cores at t and

$$R^{GDS,sys}(t,\ell) = \int \cdots \int_{\mathbf{D}} \left(R^{GDS}(t|\ell)\right)^{n-\ell} \prod_{i=0}^{\ell-1} \left((n-i)\cdot f^{GDS}(t_{i+1}|i)\right) \mathrm{d}t_1 \cdots \mathrm{d}t_\ell \tag{4.19}$$

the i^{th} permanent component failure in the system occurs at time t_i ($1 \leq i \leq \ell$), its reliability can be expressed as

$$R^{GDS}(t|\ell) = \mathcal{R}\left(\theta \cdot \left(\sum_{i=0}^{\ell-1} \Omega_{n-i} \cdot (t_{i+1} - t_i) + \Omega_{n-\ell} \cdot (t - t_\ell)\right)\right) \tag{4.18}$$

The event that all surviving components after ℓ core failures is still functioning at time t, where $t > t_\ell$, can be modeled as a series failure system. We use $\left(R^{GDS}(t|\ell)\right)^{n-\ell}$ to represent its probability. The next step is to uncondition it by being aware that it is conditioned, that the occurrence time of past failures are assumed to be given. Similar to the system lifetime analysis in [46], as the reliability of a core given past i failures is $R^{GDS}(t|i)$, the event for its $(i+1)$ failure occurring at t_{i+1} has probability $-\frac{\mathrm{d}}{\mathrm{d}t} R^{GDS}(t|i)\Big|_{t=t_{i+1}}$, referred to as $f^{GDS}(t_{i+1}|i)$ hereafter. Therefore, denoting by $R^{GDS,sys}(t,\ell)$ the probability of event that the system has experienced exactly ℓ core failures before time t and by the theorem of total probability, we have Eq. (4.19) (see next page), where the domain is $\mathbf{D} = \{(t_1, \cdots, t_\ell) \in \mathbb{R}^\ell : 0 < t_1 < \cdots < t_\ell < t\}$.

Then, since the event that a GDS system is functioning can be expressed as the union of a set of mutually exclusive events, the system reliability $R^{GDS,sys}(t)$ is therefore given by

$$R^{GDS,sys}(t) = \sum_{\ell=0}^{n-k} R^{GDS,sys}(t,\ell) \tag{4.20}$$

Consequently, the system mean time to failure is Eq. (4.21).

$$MTTF^{GDS,sys} = \text{E[service life of gracefully degrading system]} = \int_0^\infty R^{GDS,sys}(t)\mathrm{d}t \tag{4.21}$$

4.4.2 Processor Rotation System (PRS)

Processor cores can be used in a rotation manner to balance their aging effect. That is, they operate alternatively in active mode and spare mode and spend a relatively longer period in each mode when compared to the execution time of each single task in every state. At the same time, the duration is quite small when compared to the lifetime of the system. To take an example, suppose a core's lifetime is measured in years and the execution time of a task ranges from a few seconds to a few minutes, the duration for cores in each state can be several days, and reconfiguration is performed afterwards to wake up some spare cores to replace a few active cores. In [90], the authors showed an example for caches enabled in a round-robin manner.

For modeling lifetime reliability, we consider a more general case that in any configuration k out of n cores serve as active ones while the remaining $(n-k)$ have no power supply. The reconfiguration is conducted every time interval T_r, which is much shorter than a core's service life (typically a few years) but much longer than a task's execution time. At every reconfiguration, the $(n-k)$ oldest cores (that is, the cores with highest age) are shut down, and all spare ones convert to active mode. From a core's point of view, before the first failure in the system, its accumulated time up to time t as active core can be approximated as $\frac{k}{n} \cdot t$. Its wear-out rate under this condition should be Ω_k, because there are k active cores sharing workload. On the other hand, a core does not have power supply in the remaining $(1-\frac{k}{n}) \cdot t$. Recall that the wear-out rate in these time intervals is approximated to zero. Therefore, its reliability before the first component failure is given by

$$R^{PRS}(t|0) = \mathcal{R}(\theta \cdot \Omega_k \cdot \frac{k}{n} \cdot t) \tag{4.22}$$

Then, we generalize this expression to the case that the number of failure cores in the system can be 0, 1, $\cdots$, $(n-k)$. From t_i to t_{i+1}, the system composed of $(n-i)$ good components within which k are active at any time. Since a surviving core's accumulated time in this time interval depends on the quantity of both active cores and redundant ones, it can be approximated as $\frac{k}{n-i} \cdot (t_{i+1} - t_i)$. Its wear-out rate, on the other hand, remains Ω_k. We therefore compute its reliability by

$$R^{PRS}(t|\ell) = \mathcal{R}\left(\theta \cdot \Omega_k \cdot \left(\sum_{i=0}^{\ell-1} \frac{k}{n-i} \cdot (t_{i+1} - t_i) + \frac{k}{n-\ell} \cdot (t - t_\ell)\right)\right) \tag{4.23}$$

The sequential analysis is very similar to that of gracefully degrading systems (Section 4.4.1) and hence omitted here.

4.4.3 Standby Redundant System (SRS)

In SRS, k-out-of-n cores are initially configured as active units, while the remaining $(n-k)$ cores are in spare mode. Upon detection of a permanent component failure, the system attempts to wake up a spare core and configure it as an active one. Note that, different from the strategy of PRS which aims to balance the age of all cores, in SRS only when some active cores fail, cold standbys might convert into active mode, which will lead to significant difference between cores in terms of age. For example, suppose the first core failure occurs when the system has been used for 4 years, after reconfiguration the system will be composed of $(n-1)$ 4-year old cores and a brand-new one.

Consider a core that starts its service life at time t^s. From t^s to its failure or the entire system's failure, its wear-out rate is a constant Ω_k, because the quantity of active cores in the system is always k and this core is always one of them. As a

result, its reliability only depends on its service time up to t while is independent of the failures in the systems, given by

$$R^{SRS}(t,t^s) = \mathcal{R}(\theta \cdot \Omega_k \cdot (t-t^s)) \tag{4.24}$$

To evaluate $MTTF^{SRS,sys}$, it is necessary to compute the probability that all surviving cores after ℓ failures are still operational at time t. It can be expressed by considering u_i (the quantity of surviving cores starting their service life from time t_i), i.e., $\prod_{i=0}^{\ell} \left(R^{SRS}(t,t_i)\right)^{u_i}$. Note that, u_i is function of past failure history h. As this event has a condition that h occurs, we can express $P(h)$ the probability of history h according to [46]. Let $\mathcal{H}$ be the set of all possible histories. According to the theorem of total probability, the unconditional reliability is therefore

$$R^{SRS,sys}(t,\ell) = \sum_{h\in\mathcal{H}} \int \cdots \int_{\mathbf{D}} \prod_{i=0}^{\ell} \left(R^{SRS}(t,t_i)\right)^{u_i(h)} \cdot P(h)\mathrm{d}t_1 \cdots \mathrm{d}t_\ell \tag{4.25}$$

whose domain **D** is as same as that of Eq. (4.19). Similar to the analysis of gracefully degrading system, the expected service life is given by

$$MTTF^{SRS,sys} = \int_0^{\infty} R^{SRS,sys}(t)\mathrm{d}t = \int_0^{\infty} \sum_{\ell=0}^{n-k} R^{SRS,sys}(t,\ell)\mathrm{d}t \tag{4.26}$$

4.4.4 Extension to Heterogeneous System

Up to now, we have shown how to model the lifetime reliability of multi-core systems with various redundancy configurations, wherein we we regard the entire multi-core processor as a k-out-of-n: G That is, the system is initially composed of n cores and it fails when the total amount of good cores is less than k. In practice, some designs may consist of more than one type of processor cores [44]. For example, Cell processor [55] contains a high-performance PowerPC processor element (serving as main processor) and eight synergistic processor elements (as co-

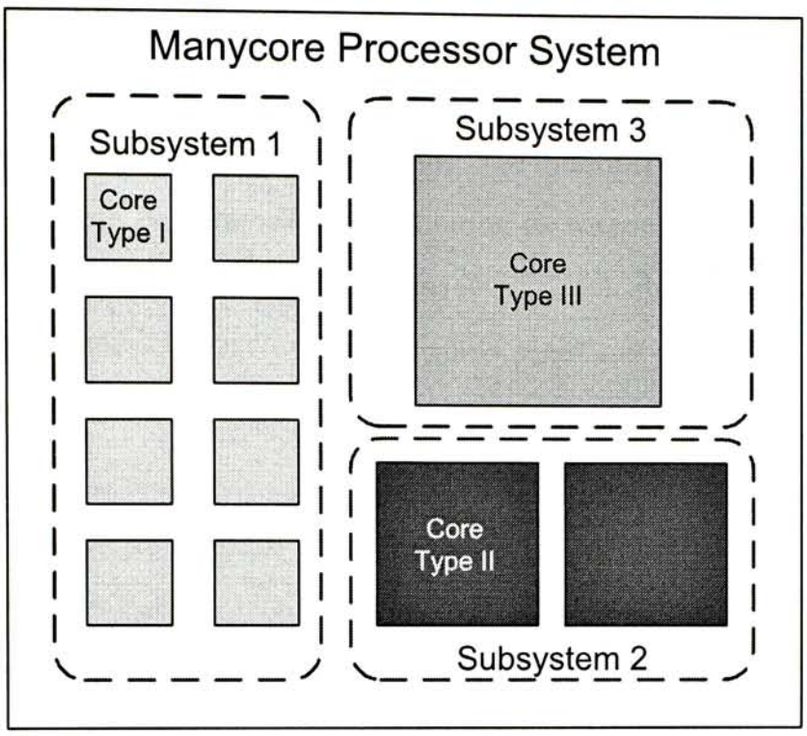


Figure 4.2: An Example Heterogeneous Multi-core Processor.

processors). It is expected that, in such a system if the main processor and 7-out-of-8 co-processors have not suffered from permanent faults, the entire system is operational. This event can be modeled by simply extending our model presented in the previous sections. That is, since we should make sure the functionality of the sole main processor, it can be specified as 1-out-of-1: G subsystem. On the other hand, the co-processors can be treated as a 7-out-of-8: G subsystem, whose configuration scheme can be arbitrary, such as standby redundant and gracefully degrading. Thus, assuming the failures within the two subsystem are independent, the lifetime reliability of the entire system comes down to the probability that both subsystems are operational.

More generally, consider a system that can be divided into q subsystems, in which subsystem i ($1 \leq i \leq q$) contains n_i identical components initially and functions if no less than k_i are operational. An example is shown in Fig. 4.2. Each

subsystem can has its own redundancy configuration scheme, referred as CFG_i in the superscript. Because of their different functionalities, we assume cores from different subsystems do not share workloads. The lifetime reliability of subsystem i at time t can then be obtained by substituting its parameters n_i and k_i into the models presented in Section 4.4, denoted as $R^{CFG_i,sys}(t)$. Recall that the functioning of all subsystems is essential for the entire system to operate properly. The expected service life of the entire system is hence given by

$$MTTF^{HS,sys} = \int_0^\infty \prod_{i=1}^{q} R^{CFG_i,sys}(t)\mathrm{d}t \tag{4.27}$$

4.5 Experimental Methodology

To compare multi-core systems with different redundancy configurations, we conduct extensive experiments on a 36-core processor (i.e., $n = 36$) with various workloads, with the number of active cores k ranging from 32 to 36. We implement a discrete event simulator to perform task allocation and scheduling for an application flow composed of 60,000 tasks in every experiment and we generate the associated power trace files for the entire system accordingly. Next, we take these files as the input of HotSpot tool [92] to acquire the temperature trace files. All temperature samples are collected to extract the temperature distribution (as in Fig. 4.1). We then compute the wear-out rate Ω according to its definition, and finally obtain the lifetime reliability of multi-core systems with various redundancy configurations, by computing multidimensional integral with Monte Carlo simulation.

4.5.1 Workload Description

As discussed earlier, workloads determine the temperature distribution of the processor. In our experiments, we generate a task flow for each workload, which is characterized by the task interarrival time distribution and the task service time distribution.

We assume the task interarrival time is with an exponential distribution with rate λ. Assuming that all the given m active cores equally share the workload, the task interarrival time of a core is $\frac{\lambda}{m}$. The task service time is modeled as exponential distribution and bimodal hyperexponential distribution. The exponential distribution is widely-used in the literature, while the bimodal hyperexponential distribution is regarded as the most probable distributions for modeling processor service time [106]. Exponential distribution has the expected service rate μ. Bimodal hyperexponential distribution is composed of two exponential distributions with mean $\frac{1}{\mu_1}$ and $\frac{1}{\mu_2}$ respectively, where $\frac{1}{\mu_1} = \frac{1}{\mu} + \frac{1}{\mu}\sqrt{\frac{(C_x^2-1)\alpha}{2(1-\alpha)}}$ and $\frac{1}{\mu_2} = \frac{1}{\mu} - \frac{1}{\mu}\sqrt{\frac{(C_x^2-1)(1-\alpha)}{2\alpha}}$. We set $\alpha = 0.95, C_x = 3.0$ [113]. For both distributions, the system load is defined as $\rho^{sys} = \frac{\lambda}{\mu}$. Consequently, each active core's load is $\rho = \frac{\lambda}{m\mu}$.

4.5.2 Temperature Distribution Extraction

In our experiments, the die size of each core is set to be $5.76mm^2$ ($2.4mm \times 2.4mm$). Depending on its current workload, an active core can be in one of two states: *Run* and *Idle*. To represent the uneven power densities in the processing unit, a core contains a small block (e.g., Execution Unit) with higher power density, whose size is $0.5mm \times 0.5mm$. The power density values of this hotspot block and other parts in *Run* state are $5.0W/mm^2$ and $0.5W/mm^2$, while that in *Idle* state are both $0.16W/mm^2$. These system parameters are set according to state-of-the-art processors (e.g., IBM PowerPC 750CL [56]). The standbys are assumed to be

in *Shut Down* state, whose power consumption is $\sim 0W$.

4.5.3 Reliability Factors

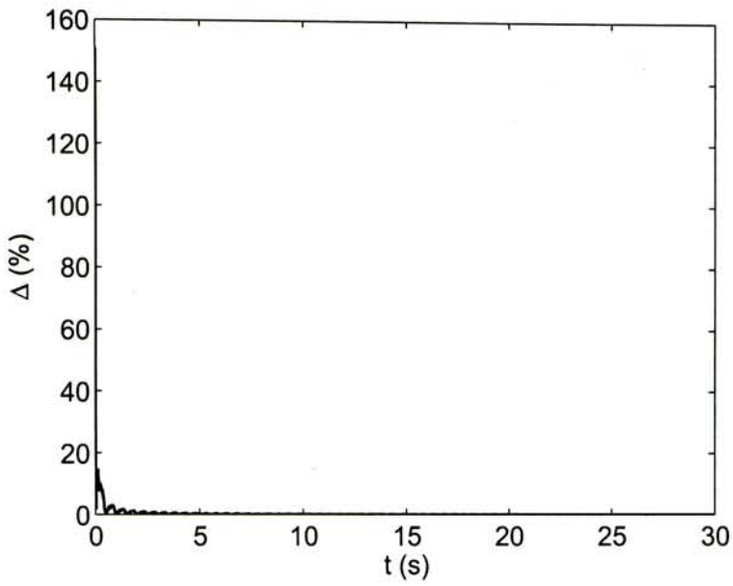
We use Weibull distribution, a well-accepted lifetime distribution for modeling hard errors of IC product [2], as the reliability function used in our system, $\mathcal{R}(t) = e^{-(\frac{t}{\theta})^{\beta}}$. shape parameter $\beta = 2.5$. Although the proposed approach is applicable for any failure mechanisms or their combinations, due to the lack of public empirical data on the relative weights of different failure mechanisms on real circuits, we analyze the electromigration failure in our experiments, whose models is presented in [39].

To compare the systems' lifetimes in various configurations, we normalize them to a certain scenario, wherein all cores of a 36-core system without redundancy are in active mode and the system workload is 5.0 and its Ω is set to be 0.1. In other words, a core in such a system has expected service life of 10 years.

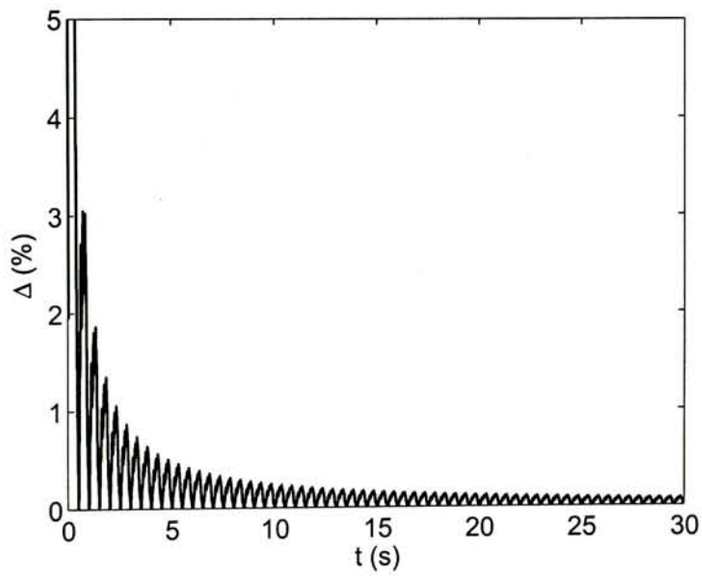
4.6 Results and Discussions

4.6.1 Wear-out Rate Computation

We first demonstrate the effectiveness for one of the key concepts in this work, the wear-out rate Ω computation, with experiments. On one hand, we trace the temperature variation of a core for 15,000 steps after the system has been warmed up, each corresponding to $3.33\mu s$, from which the temperature distribution $\nu(T)$ is extracted. The wear-out rate Ω is then computed according to Eq. (4.15). From Eq. (4.14), the component reliability can be expressed as a function of $t \cdot \Omega$ (refer to as T_Ω hereafter). We compute T_Ω for 30 seconds. On the other hand, the temperature variation of the same core is traced for 30 seconds (around 9×10^6 steps)



(a) Original Scale



(b) Enlarged Scale

Figure 4.3: The Effectiveness of Wear-out Rate Approximation.

for comparison. We use T_{trace} to represent the summation of $\frac{\Delta t}{\theta_s(T,V,f)}$ up to time t, where $\Delta t = 3.33\mu s$. The difference between T_Ω and T_{trace} versus time t is shown in Fig. 4.3.

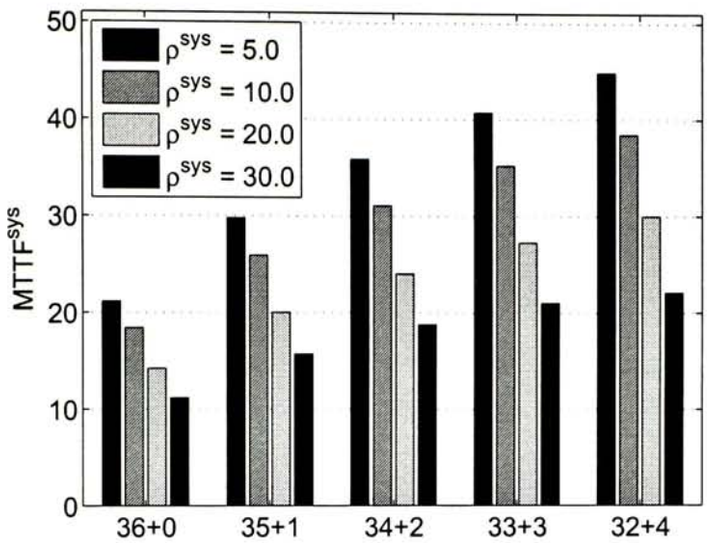
As can be seen from this figure, only in the first $4.832s$, the difference between

the approximated Ω value and the actual value is larger than 0.5%. After that, T_Ω becomes very close to T_{trace}. Since the service life of processors is typically in the range of years, the estimation error by the proposed approach is negligible.

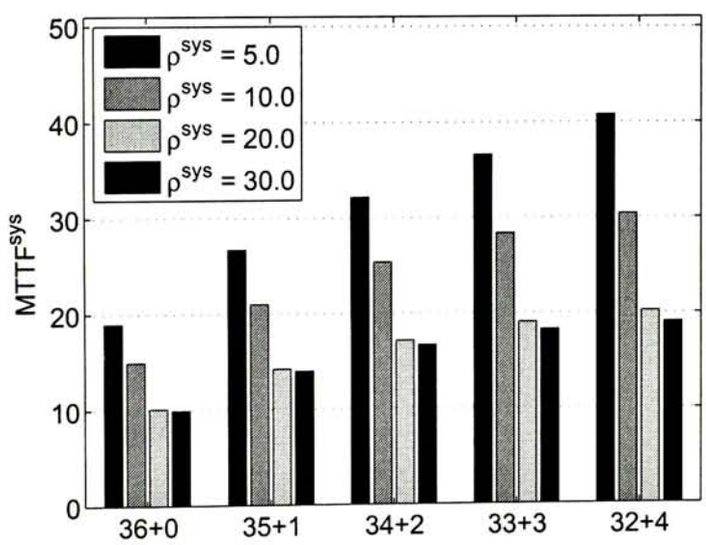
4.6.2 Comparison on Lifetime Reliability

Fig. 4.4 shows one of the most important metrics reflecting system lifetime, mean time to failure, under various redundancy configurations and workloads. With the increase of redundant cores, it is expected to have system lifetime extension and all the subfigures show this trend. Also, a closer observation of these figures show that the lifetime growth rate becomes smaller when more cores are configured as redundancy, i.e., the sojourn time of i-Failure state is larger than that of $(i+1)$-Failure state (see Fig. 4.5). This is mainly due to the increasing failure rate of IC products. Consider three PRS systems with 0, 2, and 4 redundant cores as an example (the three middle bars in Fig. 4.5). From 36+0 to 34+2 systems, the sojourn time in 0-Failure state rises from 21.16 to 22.35, increased by 1.19, while the additional 1-Failure and 2-Failure states for the 34+2 system provide 8.87 and 6.35 extra service life, respectively. As a result, the overall lifetime extension is 16.41. If we further increase the number of redundant cores by two, the lifetime extension is 12.25, less than 16.41. From the above, improving the lifetime reliability of multi-core processors by increasing the value of k gradually diminishes and it may not quite beneficial to set it as a very large value.

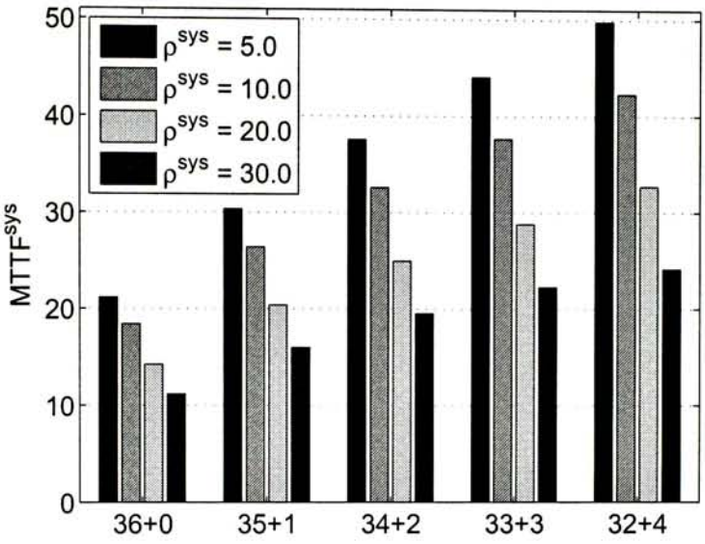
From Fig. 4.4 and Fig. 4.5, we can also see that PRS provides longer service life than the other two configurations under the same workloads. On one hand, when compared to standby systems, processor cores in PRS have a more balanced workload. That is, in SRS, some cores are set as cold standbys initially and convert to active mode only when some active cores fail. Thus, even if the workload is evenly distributed among all active cores, after some replacements the system is



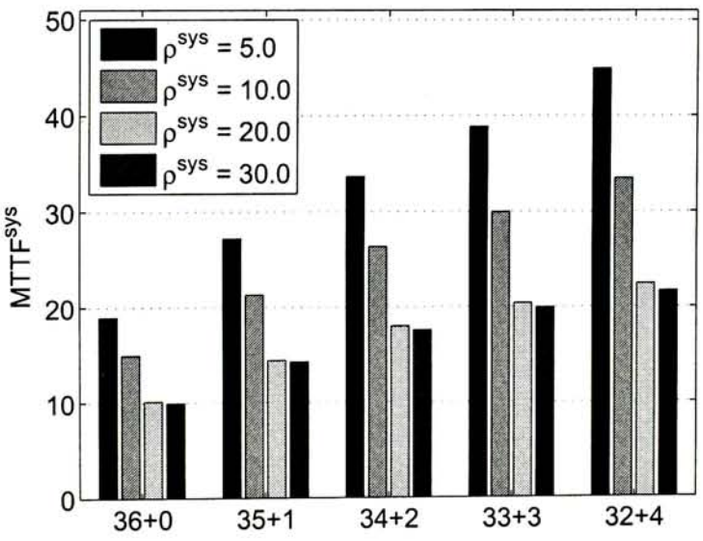
(a) Gracefully Degrading System, Exponential



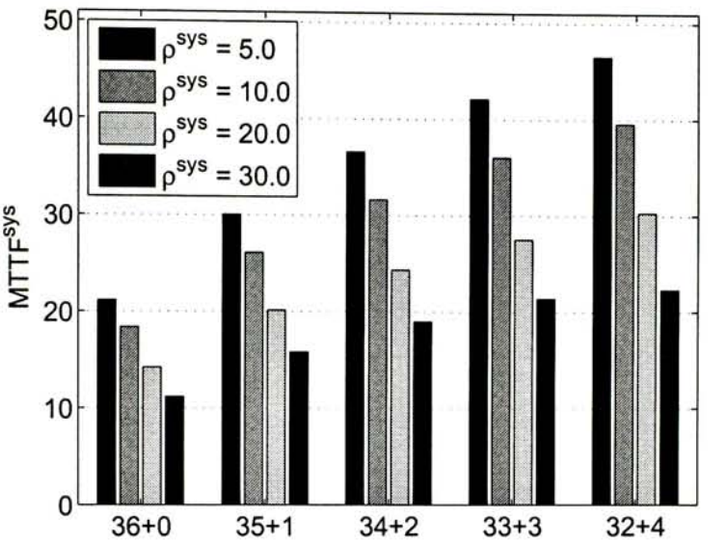
(b) Gracefully Degrading System, Bimodal Hyperexponential



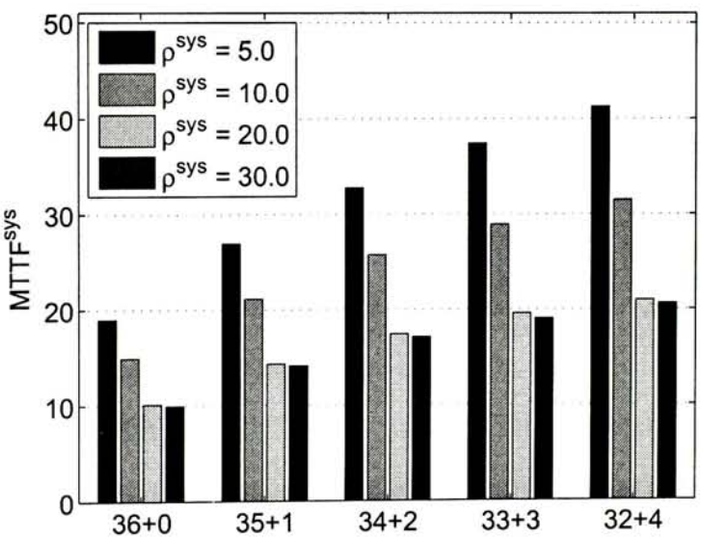
(c) Processor Rotation System, Exponential



(d) Processor Rotation System, Bimodal Hyperexponential



(e) Standby Redundant System, Exponential



(f) Standby Redundant System, Bimodal Hyperexponential

Figure 4.4: Comparison of Different Redundancy Configurations under Various Workload.

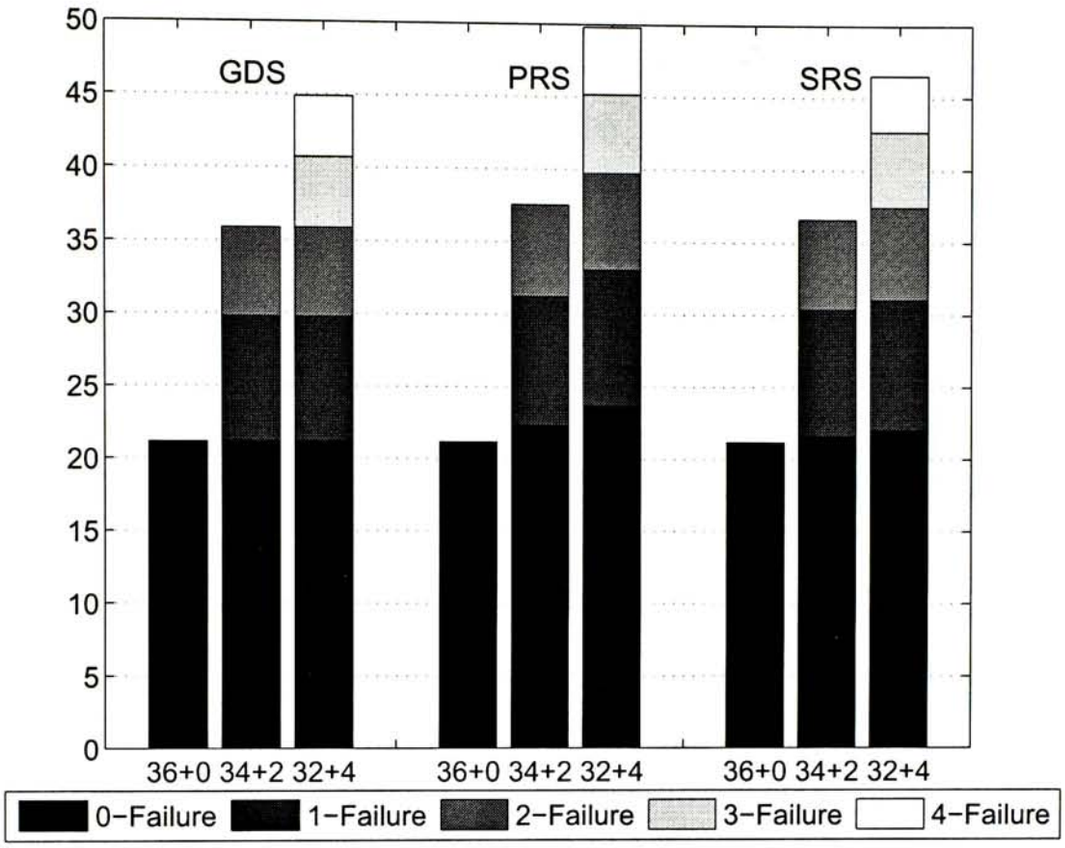


Figure 4.5: Detailed Sojourn Time in Various States.

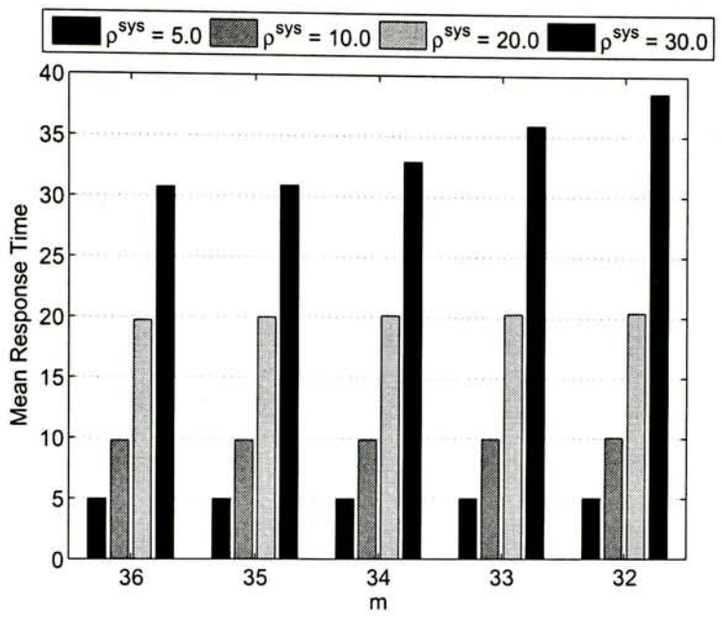
composed of many old components and a few new ones. Since the aged cores have already had high failure rate, although there are some new cores, the lifetime of the entire system cannot be extended much. From this aspect, a lot of potential computation capabilities of standbys are wasted. This problem can be avoided by using PRS configuration, which aims to balance the aging effect among all cores. On the other hand, when compared to gracefully degrading systems, processor cores in PRS alternate between active and standby modes while all cores in GDS keep aging in its lifetime. Although PRS can result in slightly heavier workload on every single core than GDS, the extra aging effect because of this issue is quite small when k is much smaller than n.

4.6.3 Comparison on Performance

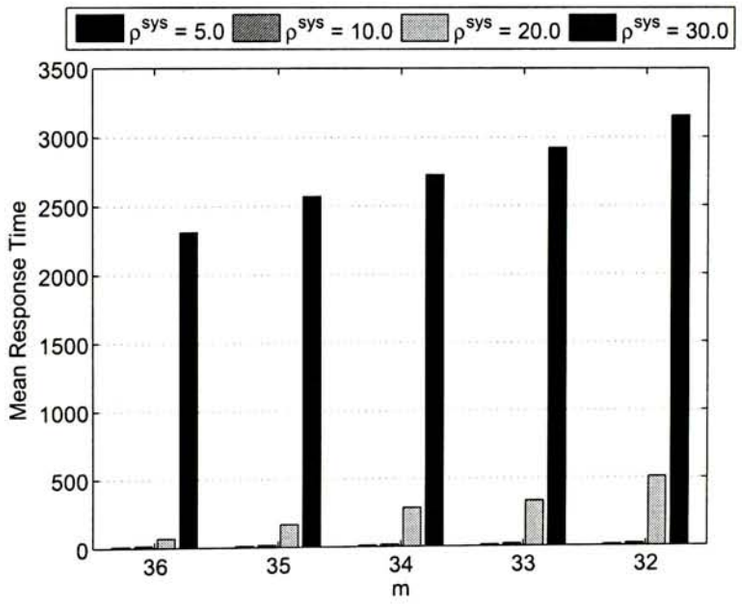
The various redundancy configurations also result in different performances for the multi-core processors. Two widely-used metrics, *mean response time* and *system utilization*, defined as the expected time from a task's arrival until its completeness and the average percentage of cores under-usage over time, respectively, are used to evaluate the performances of the multi-core systems. The results are achieved by using the same discrete-event simulator.

Fig. 4.6 shows the mean task response time versus the number of active cores in the system under various workloads. Consider exponential service time first (Fig. 4.6(a)). When the workload is not high ($\rho^{sys} \leq 20.0$), the mean response time slightly increases with the decline number of active cores. In addition, this value roughly doubles as the workload becomes twice larger. For instance, when all 36 cores serve as active ones, the mean response times corresponds to ρ^{sys} =5.0, 10.0 and 20.0 are 4.96, 9.79, and 19.73, respectively. When the workload is high (i.e., $\rho^{sys} = 30.0$), the mean response time is still roughly proportional to workloads, but a few less active cores lead to a noticeable increment of response time. For bimodal hyperexponential service time (Fig. 4.6(b)), while the systems with $\rho^{sys} = 30.0$ have similar lifetime with $\rho^{sys} = 20.0$ (see Fig. 4.4), their mean response time is significantly larger (ranging between $6.5 - 33.4\times$). We attribute this phenomenon to the close-to-saturated system workload under such circumstances. In other words, with the parameters setup in our experiments, when the workload of systems with bimodal hyperexponential distribution becomes larger than 20.0, almost all active cores always have tasks to perform, thus leading to the dramatic increase of the mean response time of tasks.

When it comes to the performances of various redundancy configurations, a GDS system sequentially has 36, 35, $\cdots$ active cores in its lifetime and therefore experiences gracefully degrading performance from the users' point of view. Other



(a) Exponential



(b) Bimodal Hyperexponential

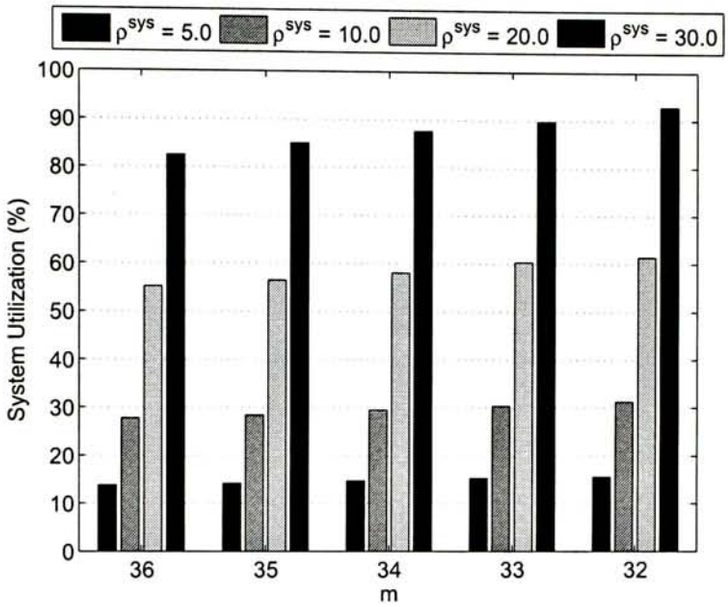
Figure 4.6: Mean Response Time under Various Workload.

configurations, by contrast, are with the same performance over its lifetime. For example, a PRS/SRS system that has 32 active cores at any time always provides mean task response time 38.60, given exponential service time and $\rho^{sys} = 30.0$. A 32+4 GDS system, however, is able to provide better performance in the first several years (its mean response time is 30.75), and then gradually increase to 30.89, 32.90, 35.89, and finally 38.60.

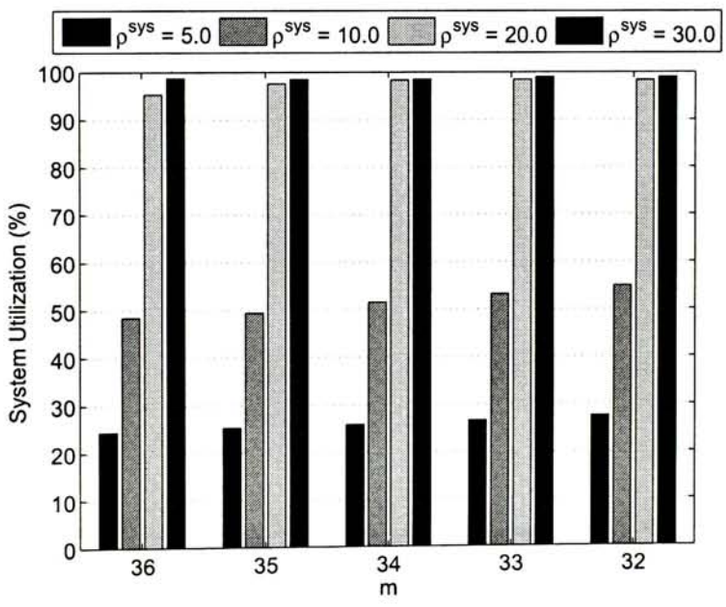
Fig. 4.7 shows the system utilization in various cases. For exponential service time, the system utilization is almost proportional to their system workload with the number of active cores. Under the fixed workloads, we can also observe slightly higher system utilization for systems with less active cores. When the workload becomes sufficiently heavy with hyperexponential distributed service time (when $\rho^{sys} \geq 20.0$), the system utilization increases very little with the increment of workloads. This can well explain the mean response time shown in Fig. 4.6(b).

4.6.4 Comparison on Expected Computation Amount

In this subsection, we combine the performance and lifetime reliability into a unified metric, namely *expected computation amount*, which reflects the amount of computation performed by a system before its failure. The results for 32+4 and 34+2 systems are shown in Fig. 4.8. An interesting phenomenon can be observed from these figures. That is, as the system workload becomes heavier, in spite of significant decline in the system lifetime (see Fig. 4.4), in most cases the total computation amount of the system increases. This is mainly because, although the system with light workload has relatively lower temperature when compared with that with heavy load, the induced difference in lifetime is less than the difference in system utilization. In particular, consider GDS shown in Fig. 4.8(a) as an example. The sojourn time in 0-Failure states for $\rho^{sys} = 5.0$ and 10.0 is

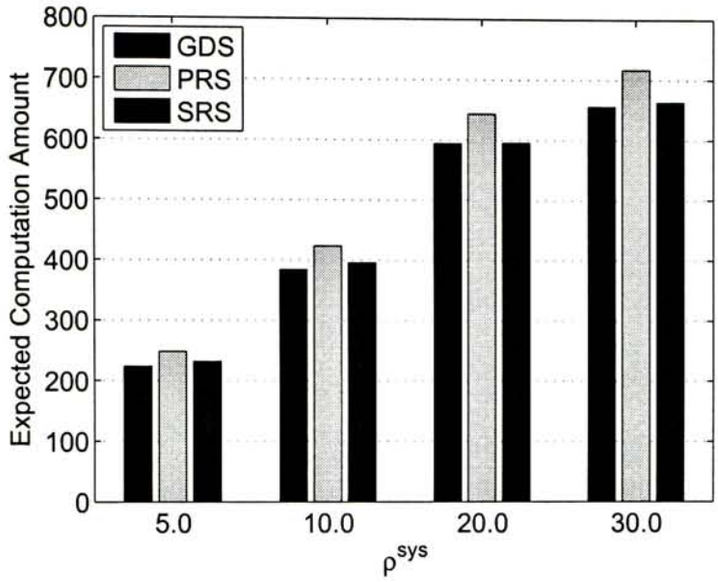


(a) Exponential

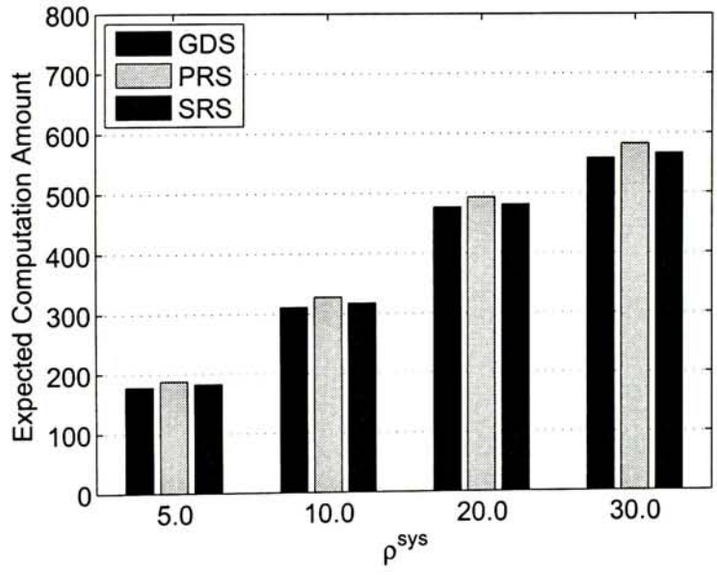


(b) Bimodal Hyperexponential

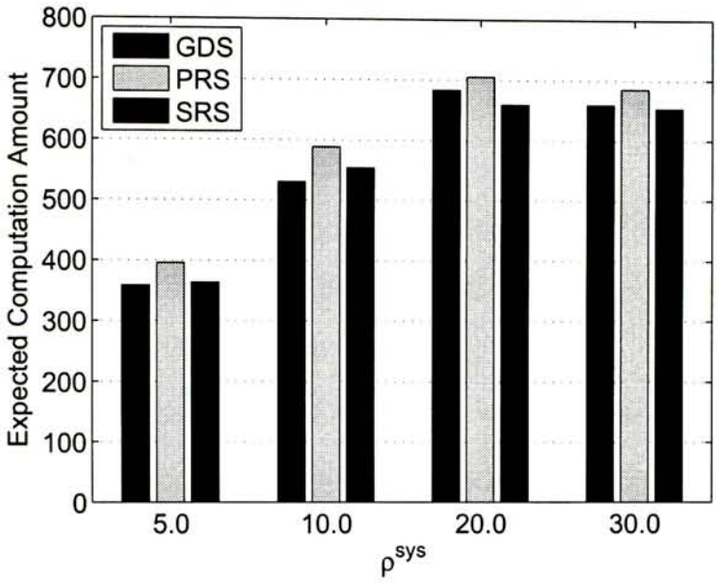
Figure 4.7: System Utilization under Various Workload.



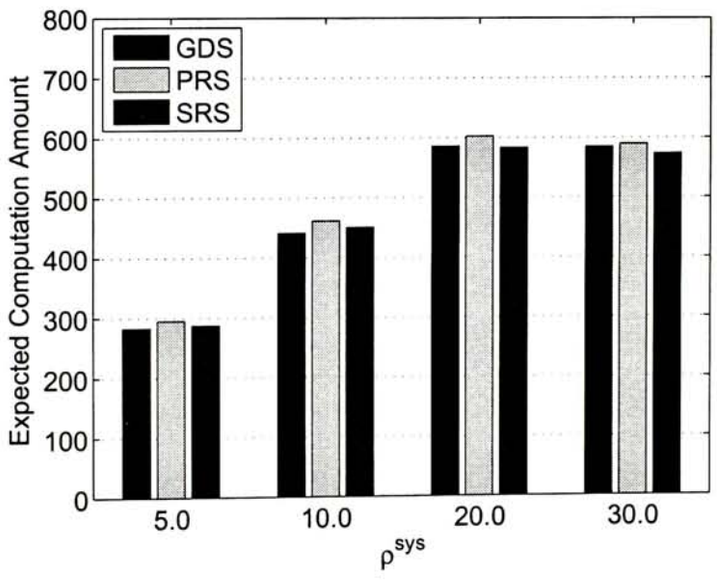
(a) Exponential, 32+4 System



(b) Exponential, 34+2 System



(c) Bimodal Hyperexponential, 32+4 System

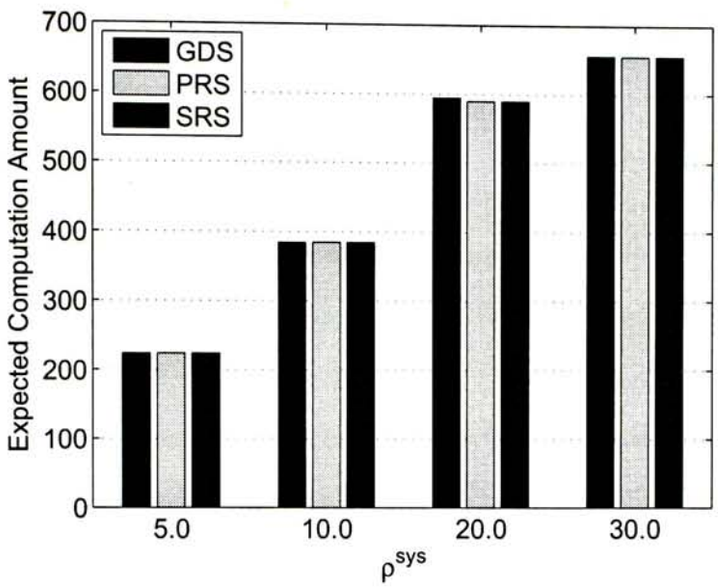


(d) Bimodal Hyperexponential, 34+2 System

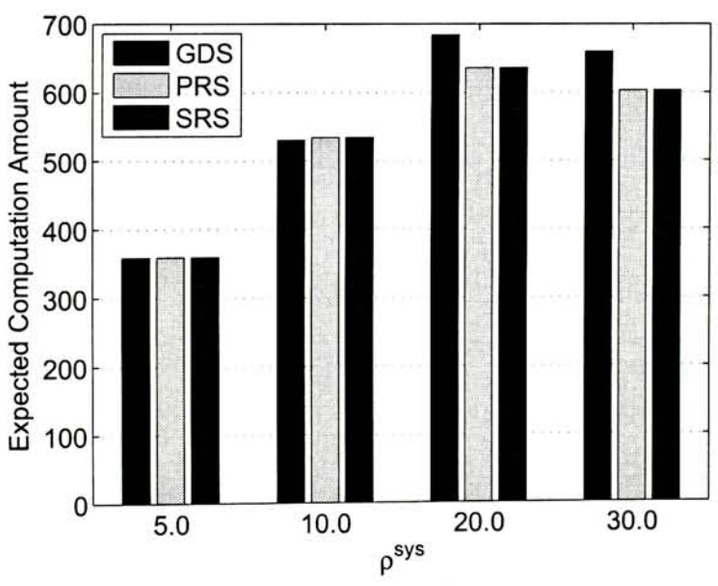
Figure 4.8: Expected Computation Amount Before System Failure.

21.16 and 18.42, respectively, while the system utilization of two cases is 13.80% and 27.76%. Therefore, the expected computation amount ratio in this state is around 1.75, and we can observe similar trends in other states. From this aspect, longer service life does not mean more effective usage of the multi-core processor. Moreover, we notice that when ρ^{sys} of the system with bimodal hyperexponential service time distribution increases from 20.0 to 30.0, the expected computation amount decreases (see Fig. 4.8(c)-(d)). The main reason lies in the fact that both cases nearly make full use of their resources and thus their computation amounts are mainly bounded by their service lives.

In some cases, we may not use computer systems until the end of their lifetimes. Hence, we are also interested in the computation amount of systems under such situations. In the following experiments, we set the minimum expected service life among GDS, PRS, and SRS computed by the proposed model as the actual system service life, and calculate the expected computation amount until that time point for the three redundancy configurations. The results for 32+4 systems are shown in Fig. 4.9. When the systems are not fully used (i.e., exponential service time with $\rho^{sys} = 5.0$, 10.0, 20.0, and 30.0 and bimodal hyperexponential with $\rho^{sys} = 5.0$ and 10.0), we can see that the total computation amounts with different configurations are the same. This is because, as the system is not fully utilized, it is always able to complete tasks within a very short time period (compared to the system's lifetime). Therefore, the computation amount equals to the task amount feeded to the system. If the system has sufficient high utilization (i.e., hyperexponential distributed service time with $\rho^{sys} = 20.0$ and 30.0 in Fig. 4.9(b)), GDS systems finish more jobs than the other two configurations. This is expected because GDS systems contain more active units and keep them busy, leading to greater computation amount. It is also worth to note that with the same workload distribution, PRS and SRS always have the same computation amount when con-



(a) Exponential



(b) Bimodal Hyperexponential

Figure 4.9: Comparison of Three Redundancy Configurations in Expected Computation Amount with the Same Service Time.

sidering the same service time. This is because in both configurations the number of active cores at any time remains the same (32, in our experiments).

4.7 Conclusion

In this paper, we propose a novel analytical model to characterize the lifetime reliability of multi-core processors with various redundancy configurations. Our proposed model is able to capture the impact of temperature variations of processor cores and workloads. Our experiments compare the lifetimes and performances of gracefully degrading systems, processor rotation systems and standby redundant systems, under various workloads.

□ End of chapter.

Part III

Applications

Chapter 5

Task Allocation and Scheduling for MPSoCs

The content of this chapter is included in the proceedings of *IEEE/ACM Design, Automation, and Test in Europe (DATE)* 2009 [52] and has been accepted for publication in *IEEE Transactions on Parallel and Distributed Systems* [53].

5.1 Introduction

As technology advances, it is possible to integrate multiple microprocessors, dedicated hardware accelerators, and sometimes mixed-signal circuitries on a single silicon die, namely multiprocessor system-on-chip [60]. One way to design MPSoC embedded systems is to use hardware/software co-synthesis [100]. While this method is able to explore more design space to obtain a flexible application-specific architecture, it generally takes more design time and has high design risk. Because of this, platform-based design methodology has become increasingly popular for complex embedded systems. In this approach, designers first pick a predesigned MPSoC platform, e.g., ARM11 PrimeXsys platform [7] or NXP Nexpe-

ria platform [109], and then map their applications onto this platform.

While the relentless scaling of CMOS technology has brought MPSoC designs with enhanced functionality and improved performance in every new generation, at the same time, the associated ever-increasing on-chip power and temperature densities make failure mechanisms serious threats for the lifetime reliability of such high-performance integrated circuits [16, 95, 118]. If care is not taken during the task allocation and scheduling process, some processors might age much faster than the others and become the reliability bottleneck for the embedded system, thus significantly reducing the system's service life.

Although there are many existing works on reliability-driven task allocation and scheduling (e.g., [33, 87, 98, 105]), most of them assume an exponential distribution for failure mechanisms. In other words, processors' failure rates are assumed to be independent of their usage history, which is obviously inaccurate: a typical wear-out failure mechanism will have increasing failure rate as the circuit ages [46, 97]. Recently, some thermal-aware task scheduling techniques have been proposed in the literature (e.g., [111]). As ICs' failure rates are strongly related to their operational temperature, these techniques may implicitly improve the MPSoC's lifetime reliability, by balancing different processors' temperatures or keeping them under a safe threshold. However, since many other factors (e.g., internal structure, operational frequency, and supply voltage) also severely affect the circuits' failure rate [91, 95], without explicitly taking the lifetime reliability into account during the task allocation and scheduling process, processor cores may still age differently and hence result in shorter mean time to failure for MPSoC designs.

In this paper, we present novel solutions for the lifetime extension of platform-based MPSoC designs. The main contributions of our work are as follows:

- we propose a comprehensive lifetime reliability-aware task allocation and

scheduling strategy that takes processors' aging effect into account, based on simulated annealing (SA) technique;

- we present a novel analytical model to compute the lifetime reliability of platform-based MPSoCs when executing periodical tasks;
- we propose several speedup techniques to achieve an efficient MPSoC lifetime estimation with satisfactory solution quality.

The remainder of this paper is organized as follows. Section 5.2 reviews related prior work and motivates this paper. The proposed lifetime reliability-aware task allocation and scheduling strategy is presented in Section 5.3. We then introduce our analytical model for the lifetime reliability of platform-based MPSoC designs in Section 5.4. To meet the stringent time-to-market requirement, four speedup techniques for MPSoC lifetime approximation are presented in Section 5.5. Experimental results on several hypothetical platform-based MPSoC designs are presented in Section 5.6. Finally, Section 5.7 concludes this paper and points out some future research directions.

5.2 Prior Work and Motivation

5.2.1 IC Lifetime Reliability

Various failure mechanisms that could result in IC errors have been extensively studied in the literature. They can be broadly classified into two categories: *extrinsic failures* and *intrinsic failures*. Extrinsic failures, e.g., interconnect shorts/opens during fabrication, are mainly caused by manufacturing defects. Most of them are weeded out during the manufacturing test and burn-in process [22, 112]. Intrinsic failures can be further categorized into *soft errors* and *hard errors*. As

soft errors [76] caused by radiation effect do not fundamentally damage the circuit, they are not viewed as lifetime reliability threats. In this paper, we mainly consider those hard errors that are permanent once they manifest. The most representative ones include time dependent dielectric breakdown in the gate oxides, electromigration and stress migration in the interconnects, and negative bias temperature instability stresses that shift PMOS transistor threshold voltages. Many widely-accepted reliability models for the above failure mechanisms at device- and circuit-level have been proposed and empirically validated by academia and industry [15, 2, 45, 99, 115, 73], and it is shown that they were strongly related to the temperature and voltage applied to the circuit.

The above hard intrinsic failures have recently re-attracted lots of research interests, due to their increasingly adverse effect with technology scaling. Srinivasan *et al.* [95, 97] presented an application-aware architecture-level model named *RAMP* that is able to dynamically track lifetime reliability of a processor according to application behavior, where the sum-of-failure-rate (SOFR) model is used to combine the effect of different failure mechanisms. This model, however, is inherently inaccurate because it assumes a uniform device density over the chip and an identical vulnerability of devices to failure mechanisms. Later, to address this problem, Shin *et al.* [91] defined reference circuits and presented a structure-aware lifetime reliability estimation framework that takes the vulnerability of basic structures of the microarchitecture (e.g., register files, latches and logic) to different failure mechanisms into account. The above models target a single-core processor's lifetime reliability. Coskun *et al.* [29] proposed a simulation methodology to evaluate the lifetime reliability of multi-core systems, and used it to optimize the system's power management policy. For the sake of simplicity, most of the above models assumed exponential failure distributions and thus cannot capture the processors' accumulated aging effect. In addition, for the processors' opera-

tional temperatures, the above models either used the average temperature value over a period of time or tried to trace the temperature variations accurately. The accuracy of the former method is questionable, while the computation complexity for the latter case is too high to be adopted during design space exploration.

Recently, Huang and Xu [46] proposed to model the lifetime reliability of homogeneous multi-core systems using a load-sharing nonrepairable k-out-of-n:G system with general failure distributions for embedded cores, taking core-level redundancy into account. This model assumes a processor core is in one of three states (processing, wait, and spare), each corresponding to a unique albeit arbitrary failure distribution. In practice, however, the lifetime reliability of a processor core strongly depends on its operational temperature, which varies with different applications running on it even in the same state. In addition, how to obtain the failure distributions for each state is not shown in this work.

5.2.2 Task Allocation and Scheduling for MPSoC Designs

There is a rich literature on static task allocation and scheduling algorithms. Various issues have been considered, including timing constraint, communication cost, precedence relationship, reliability, static/dynamic priority, and task duplication [18, 68]. Since the problem of scheduling tasks on multiprocessors for a single objective has been proved to be an NP-complete problem, heuristic algorithms such as list scheduling [69] are widely used in the industry. To achieve better results, various statistical optimization techniques (e.g., genetic algorithm, simulated annealing, and tabu search) were also proposed to tackle this problem.

Most prior work in reliability-driven task allocation and scheduling (e.g., [33, 87, 98]) assumes processors' failure rates to be independent of their usage history. This assumption might be applicable for modeling random soft errors in IC products, but it is obviously inaccurate for the wear-out-related hard errors considered

in this work. As discussed earlier, for lifetime reliability threats, we should consider the more reasonable increasing failure rates during the task allocation and scheduling process.

Many recent studies on task scheduling for MPSoC systems aimed at balancing different processors' temperatures or keeping them under a threshold (e.g., [30, 101, 111]). These techniques might improve the system's lifetime reliability implicitly, since operational temperature has a significant impact on ICs' lifetime reliability. However, since wear-out failures are also affected by many other factors (e.g., the circuit structure, voltage and operating frequency), these thermal-aware techniques may not balance the aging effect among processors, especially for heterogeneous MPSoCs. As a result, some processors may still age faster than the others and hence result in shorter system service life. On one hand, this heterogeneity might simply come from the different processors' microarchitectures [91]. On the other hand, even for homogeneous systems, structurally-identical processors can have different reliability budgets due to process variation. That is, imperfect manufacturing process can lead to significant variation in device parameters (such as, channel length and threshold voltage) among transistors and hence reliability-related parameters among processor cores on the same die [43, 84].

Let us consider the following motivational example. Suppose we have an MPSoC platform containing two processors P_1 and P_2. The MTTF due to electromigration can be modeled as $MTTF_{EM} \propto J^{-n} e^{\frac{E_a}{kT}} \propto (V_{dd} \times f \times p_i)^{-n} e^{\frac{E_a}{kT}}$ (typically $n = 2$ [15]), where V_{dd}, f, p_i, E_a, k, and T represent the supply voltage, the clock frequency, the transition probability within a clock cycle, a material related constant, the Boltzmann's constant, the absolute temperature, respectively [31]. Suppose $f_1 = 2f_2$, i.e., the clock frequency of P_1 is twice of that of P_2, and all other parameters are the same, the lifetime of P_2 is four times of that of P_1. That is, even if we are able to balance the operational temperatures of the two processors to be

exactly the same all the time, processor P_1 will be the lifetime bottleneck of the MPSoC because it ages much faster than P_2.

From the above, we can reach to the conclusion that it is essential to *explicitly* take the lifetime reliability into consideration during the task allocation and scheduling process for MPSoC designs [52], which motivates this work. A relevant work targeting this problem was presented in [118] recently. The authors suggested to use lookup tables that fit with lognormal distribution curves to precalculate processors' MTTF, but the details are missing. In addition, their work targets the hardware/software co-synthesis design methodology, different from the platform-based MPSoC designs studied in our work.

5.3 Proposed Task Allocation and Scheduling Strategy

In this section, we formulate the lifetime reliability-aware task allocation and scheduling problem for platform-based MPSoC designs (Section 5.3.1) and we propose to use simulated annealing technique to solve this problem. The solution representation, cost function, and simulated annealing process are presented in Section 5.3.2, Section 5.3.3, and Section 5.3.4, respectively.

5.3.1 Problem Definition

As mentioned earlier, the platform-based MPSoC may be composed of non-identical processors, where the heterogeneity comes from various sources. For example, the processors might be structurally-identical but belong to different voltage-frequency islands, or they have entirely different structures. Thus, a task may consume different execution time and power on different processors. The problem studied in this work is formulated as follows: Given

- A directed acyclic task graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, wherein each node in $\mathbf{V} = \{v_i : i = 1, \cdots, n\}$ represents a task, and $\mathbf{E}$ is the set of directed arcs which represent precedence constraints. Each task i has a deadline d_i. If a task does not have deadline, its d_i is set to be ∞;

- A platform-based MPSoC embedded system that consists of a set of k processors and its floorplan;

- Execution time table $\mathbf{L} = \{t_{i,j} : i = 1, \cdots, n, j = 1, \cdots, k\}$, where $t_{i,j}$ represents the execution time of task i on processor j;

- Power consumption table $\mathbf{R} = \{r_{i,j} : i = 1, \cdots, n, j = 1, \cdots, k\}$, where $r_{i,j}$ represents the power consumption of processor j when it executes task i;

- Parameters of failure mechanisms (e.g., the activation energy for the diffusion processes E_a of electromigration) and the time-independent parameter of the corresponding failure distributions (e.g., the slope parameter β in Weibull distribution).

To determine a static periodical task allocation and schedule that is able to maximize the expected service life (or, lifetime) of the MPSoC embedded system under the performance constraint that every task finishes before its deadline.

Note that, while we mainly consider processor cores in this work because of their heavy wear-out stress, our solution can be easily extended to take other hardware resources on the MPSoC platforms into account, if necessary. In addition, as

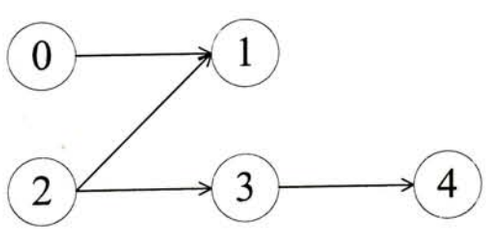


Figure 5.1: An Example Task Graph.

the first step to tackle the above complicated problem, we assume the voltage and frequency of processors do not change at runtime, although many MPSoC platforms employ dynamic voltage/frequency scaling. This work, in spite of that, is applicable for MPSoCs with multiple voltage-frequency islands.

5.3.2 Solution Representation

The edges in the task graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ indicate the dependencies of tasks, that is, there is a directed edge (v_i, v_j) in $\mathbf{E}$ if and only if task v_i must have been finished before v_j start its execution (denoted as $v_i \prec v_j$). For example, the task graph shown in Fig. 5.1 reflects the following relationship: $0 \prec 1$, $2 \prec 1$, $2 \prec 3$, and $3 \prec 4$. For any directed acyclic graph, there exists at least one order of the tasks that conforms to the partial order designated by the task graph (defined as a *valid schedule order*) and can be used as the task assignment order. For the above example, (0, 2, 1, 3, 4) and (2, 3, 4, 0, 1) are both valid schedule orders.

Thus, the task allocation and schedule for an MPSoC design can be represented as (*schedule order sequence*; *resource assignment sequence*) [80]. For example, given the task graph in Fig. 5.1 and two processors (P_1 and P_2) that can be used to execute any task, a solution represented as (0, 2, 1, 3, 4; P_1, P_1, P_2, P_1, P_2), means that task 0 is scheduled first, followed by tasks 2, 1, 3 and 4, respectively. As for the resource assignment, tasks 0, 2 and 3 are executed on P_1 while tasks 1 and 4 are assigned to P_2. Although this representation has been proposed in previous work for genetic algorithm (e.g., [80]), in this paper it is used in simulated annealing algorithm where the methodology to generate new solutions is totally different. We also provide a mathematical proof for the completeness of the search space with our proposed method (see Section 5.3.4).

Reconstructing schedule from the above solution representation is quite straightforward. In each step, we pick up a task according to the schedule order, assign

it to the corresponding processor at its earliest available time, and then update the available time of all the processors. We can then obtain the ending time of every task i (denoted as e_i) to identify whether it violates the deadline constraint d_i. Clearly, a solution corresponds to a task schedule if its schedule order conforms to the partial order defined by **G**. A possible schedule for the example solution representation is shown in Fig. 5.2.

5.3.3 Cost Function

As the guidance for decision making, cost function also plays an important role in simulated annealing. Generally speaking, the solution with lower cost means a preferable choice and hence should be accepted with higher possibility. The cost function is therefore defined following this principle. That is, in our problem, on one hand the solution should meet the performance (i.e., timing) constraints, while on the other hand we need to maximize the lifetime of platform-based MPSoC embedded systems subject to this requirement. We therefore introduce two terms into the cost function respectively as follows:

$$Cost = \mu \cdot 1_{\{\exists i: e_i > d_i\}} - MTTF^{sys} \tag{5.1}$$

where, the first term indicates the deadline violation penalty. To be specific, μ is a significant large number, and $1_{\{\cdot\}}$ is the indicator function. This function is equal to 1 if a schedule cannot meet deadline; otherwise it is equal to 0. Thus, if a schedule violates the deadline constraints, the cost of this solution will be very large and hence be abandoned. Otherwise, the first term disappears and only the second term remains.

By comparing ending time e_i and deadline constraint d_i for any task i, it is easy to know whether performance constraints are violated, while, as mentioned before, the lifetime estimation is a non-trivial problem with extremely high computational

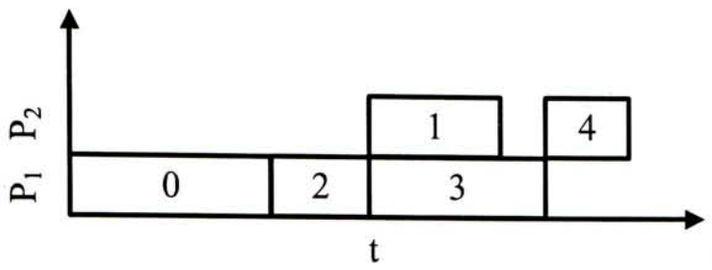


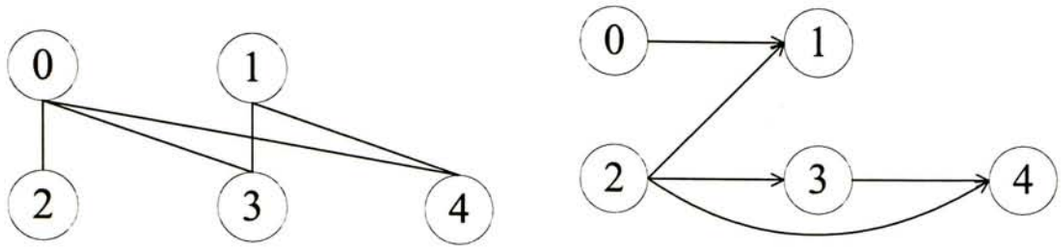
Figure 5.2: A Feasible Task Allocation and Schedule.

complexity. Our proposed method for handling this problem will be presented in Section 5.4 and 5.5 in details.

5.3.4 Simulated Annealing Process

The proposed SA-based algorithm starts with an initial solution obtained by any deterministic task scheduling algorithm (e.g., list scheduling) and the "temperature" of this solution is initialized as a high value. This temperature gradually decreases during the simulated annealing process. At each temperature T_a, a certain amount of iterations are conducted and some neighbor solutions are considered. Once we reach a new solution, its cost (denoted as $Cost_{new}$) is computed using Eq. (5.1) where $MTTF^{sys}$ is substituted by Eq. (5.26), and compared to that of the old one (denoted as $Cost_{old}$). If $Cost_{new} < Cost_{old}$, the new solution is accepted; otherwise the probability that the new solution is accepted is $e^{-(Cost_{new}-Cost_{old})/T_a}$. When T_a is as low as the pre-set ending temperature, the simulated annealing process is terminated and the solution with the lowest cost obtained so far is regarded as the final solution. During the simulated annealing process, it is important to be able to reach the entire solution space from an initial solution, in order not to fall into local minimum point.

Before introducing the details on how we identify new solutions from a random initial solution, we first introduce two transforms of directed acyclic graph. With



(a) $\widehat{\mathbf{G}}$ (b) $\widetilde{\mathbf{G}}$

Figure 5.3: Two Transforms of Directed Acyclic Graph.

the given task graph $\mathbf{G}$, we can construct an *expanded task graph* $\widehat{\mathbf{G}} = (\mathbf{V}, \widehat{\mathbf{E}})$, which has the same nodes as $\mathbf{G}$, but with more directed edges. That is, if the task graph implies a precedence constraint, an edge is added into $\widehat{\mathbf{G}}$. Fig. 5.3(a) shows the corresponding expanded task graph to the task graph in Fig. 5.1. While there is no edge (2,4) in Fig. 5.1, task 2 must be executed before task 4 because $\mathbf{E}$ contains edges (2,3) and (3,4). Thus, an edge (2,4) is included in $\widehat{\mathbf{E}}$. Moreover, we construct an undirected *complement graph* $\widetilde{\mathbf{G}} = (\mathbf{V}, \widetilde{\mathbf{E}})$. There is an undirected edge (v_i, v_j) in $\widetilde{\mathbf{E}}$ if and only if there is no precedence constraints between v_i and v_j (denoted as $v_i \circlearrowleft v_j$). The corresponding complement graph to Fig. 5.1 is shown in Fig. 5.3(b). $v_i \preceq v_j$ is used to represent that either $v_i \prec v_j$ or $v_i \circlearrowleft v_j$.

With these notations, we theoretically prove that any valid schedule order sequence of the given task graph $\mathbf{G}$ is reachable, starting from an arbitrary initial sequence, as shown in the following.

Lemma 3 *Given a valid schedule order* $A = (a_1, a_2, \cdots, a_{|\mathbf{V}|})$*, swapping adjacent nodes leads to another valid schedule order, provided there is an edge between these two nodes in graph* $\widetilde{\mathbf{G}}$.

Proof: Since A is a valid schedule order, we have the property: $a_1 \preceq a_2 \preceq \cdots \preceq a_i \preceq a_{i+1} \preceq \cdots \preceq a_{|\mathbf{V}|}$. If the edge $(a_i, a_{i+1}) \in \widetilde{\mathbf{E}}$ $(1 \leq i \leq |\mathbf{V}| - 1)$, there is no precedence constraints between them. In other words, $a_1 \preceq a_2 \preceq \cdots \preceq a_i \circlearrowleft a_{i+1} \preceq$

$\cdots \preceq a_{|\mathbf{V}|}$. If we swap the position of task a_i and a_{i+1}, no precedence constraints are violated and hence we have another valid schedule order. □

Theorem 4 *Starting from a valid schedule order* $A = (a_1, a_2, \cdots, a_{|\mathbf{V}|})$, *we are able to reach any other valid schedule order* $B = (b_1, b_2, \cdots, b_{|\mathbf{V}|})$ *after finite times of adjacent swapping.*

Proof: A feasible procedure is shown in Fig. 5.4. In this procedure, the positions of nodes in the sequence are adjusted one by one. That is, we first find node b_1 in sequence A, move it to the first position A by a series of adjacent swapping. And then find node b_2 in sequence A and move it to the second position of sequence A. After the first $(i-1)$ iterations, $a_1 = b_1$, $a_2 = b_2$, $\cdots$, $a_{i-1} = b_{i-1}$. At the i^{th} iteration, if $a_i = b_i$ there is no need to adjust the position of a_i. Otherwise, we move node a_j to the i^{th} position of sequence A by $(j-i)$ times of swapping (line 5). None of them violate precedence constraints. The main reason is: since A is a valid schedule order, we have $a_i \preceq a_{i+1} \preceq \cdots \preceq a_j$. On the other hand, since B is also a valid order, $b_i \preceq b_{i+1} \preceq \cdots \preceq b_{|\mathbf{V}|}$. Note that, $a_j = b_i$. Thus $a_i \preceq a_{i+1} \preceq \cdots \preceq a_j \preceq b_{i+1} \preceq \cdots \preceq b_{|\mathbf{V}|}$. In addition, set $\{a_i, a_{i+1}, \cdots, a_{j-1}\} \subseteq$ set $\{b_{i+1}, \cdots, b_{|\mathbf{V}|}\}$. Consequently, $a_j \circlearrowleft a_{j-1}$, $a_j \circlearrowleft a_{j-2}$, $\cdots$ $a_j \circlearrowleft a_i$. □

$(a_1, a_2, \cdots, a_{|\mathbf{V}|}) \Rightarrow (b_1, b_2, \cdots, b_{|\mathbf{V}|})$

```
1  For i = 1 to n − 1
2      If a_i ≠ b_i
3          find j, where a_j = b_i
4          For k = j − 1 to i
5              swap a_k and a_{k+1} in sequence A
```

Figure 5.4: Swapping Procedure.

Accordingly, three moves are introduced to reach all possible solutions, starting with an arbitrary valid initial solution.

- M1: Swap two adjacent nodes in both schedule order sequence and resource assignment sequence, provided that there is an edge between these two nodes in graph $\widetilde{\mathbf{G}}$.
- M2: Swap two adjacent nodes in resource assignment sequence only.
- M3: Change the resource assignment of a task.

With the above moves, all possible task schedules are reachable starting from an arbitrary initial one. This is because, for a certain resource and task binding M1 essentially can visit all other valid schedule orders starting from an initial one, while M2 and M3 guarantee that all resource assignment sequence can be tried.

In the following two sections, we present how to efficiently obtain $MTTF^{sys}$ in Eq. 5.1, i.e., the MTTF of an MPSoC design with a particular task allocation and schedule.

5.4 Lifetime Reliability Computation for MPSoC Embedded Systems

The well-accepted failure mechanism models in the literature (e.g., [2]) typically provide the relationship between $MTTF$ and a fixed temperature T. However, processors' operational temperature varies significantly with different applications. Generally speaking, when a processor is under usage or its "neighbors" on the floorplan are being used, its temperature is relatively higher than otherwise. In this section, we introduce a novel analytical method to estimate the lifetime reliability of MPSoC embedded systems running periodical tasks, within which the existing

failure models are taken as inputs and the influence of temperature variation caused by task alternations is reflected.

We use Weibull distribution to describe the wear-out effect, as suggested in JEP85 [1]. Since the slope parameter is shown to be nearly independent of temperature [23], the reliability of a single processor at time t can be expressed as

$$R(t,T) = e^{-(\frac{t}{\alpha(T)})^{\beta}} \tag{5.2}$$

where T, $\alpha(T)$, β represent temperature, the scale parameter, and the slope parameter in the Weibull distribution, respectively. Instead of assuming T as a fixed value, we consider the temperature variations in our analytical model for more accuracy. At the same time, it is important to note that the other factors that affect a processor's lifetime reliability are also considered in the model. That is, the architecture properties of processor cores are reflected on the slope parameter β, while the cores' various operational voltages and frequencies manifest themselves on $\alpha(T)$ (see Eq. (5.6)). Since temperature T varies with respect to time t, it can be regarded as a function of t. This allow us to eliminate the notation T from $R(t,T)$ in the rest of this paper.

In general, mean time to failure is defined as

$$MTTF = \int_0^{\infty} R(t)\mathrm{d}t \tag{5.3}$$

Considering Weibull distribution given in Eq. (5.2), this equation can be deduced into the formation shown below [34]

$$MTTF(T) = \alpha(T)\Gamma(1+\frac{1}{\beta}) \tag{5.4}$$

Rearranging the equations yields the expression of the scale parameter, i.e.,

$$\alpha(T) = \frac{MTTF(T)}{\Gamma(1+\frac{1}{\beta})} \tag{5.5}$$

Our analytical framework takes the hard error models as inputs, and hence it is applicable to analyze any kinds of failure mechanisms, including the combined failure effect shown in [95, 97]. For the sake of simplicity, we take electromigration failure mechanism as an example. By substituting its lifetime model into Eq. (5.5), we obtain the corresponding scale parameter

$$\alpha(T) = \frac{A_0(J-J_{crit})^{-n}e^{\frac{E_a}{kT}}}{\Gamma(1+\frac{1}{\beta})} \tag{5.6}$$

where A_0 is a material-related constant, $J = V_{dd} \times f \times p_i$ [31], and J_{crit} is the critical current density.

Depending on a processor's temperature variations with respect to time, we obtain a subdivision of the time $[0,t]$: $0 = t_0 < t_1 < t_2 < \cdots < t_m = t$. For all $\varepsilon_T > 0$ there exists $\delta_t > 0$ such that, if the largest partition $\max_i(t_{i+1} - t_i) < \delta_t$ then for all i the difference between the highest temperature in this interval $\max_{t_i<\tau<t_{i+1}} T(\tau)$ and the lowest one $\min_{t_i<\tau<t_{i+1}} T(\tau)$ is less than ε_T. Denote by $[t_i, t_{i+1})$ the $(i+1)^{\text{th}}$ time interval and let $\Delta t_i = t_{i+1} - t_i$. We assume that the temperature during $[t_i, t_{i+1})$ is an arbitrary constant T_i within the range $[\min_{t_i<\tau<t_{i+1}} T(\tau), \max_{t_i<\tau<t_{i+1}} T(\tau)]$. We know that the initial reliability of the processor is given by

$$R(t)\big|_{t=t_0} = 1 \tag{5.7}$$

For the first interval $[t_0, t_1)$, since the temperature is fixed to T_0, by Eq. (5.2), we have

$$R(t) = e^{-(\frac{t}{\alpha(T_0)})^{\beta}}, \qquad t_0 \le t < t_1 \tag{5.8}$$

At the end of this interval, the reliability is

$$R(t_1^-) = e^{-(\frac{t_1}{\alpha(T_0)})^\beta} \tag{5.9}$$

Using a quantity c to represent the aging effect in $[t_0, t_1)$, we express the reliability in the second interval as

$$R(t) = e^{-(\frac{t+c}{\alpha(T_1)})^\beta}, \qquad t_1 \leq t < t_2 \tag{5.10}$$

At the beginning of the second interval

$$R(t_1^+) = e^{-(\frac{t_1+c}{\alpha(T_1)})^\beta} \tag{5.11}$$

c can be computed by the continuity of reliability function, that is, $R(t_1^-) = R(t_1^+)$, yielding

$$c = (\frac{\alpha(T_1)}{\alpha(T_0)} - 1) \cdot t_1 \tag{5.12}$$

Substituting it into Eq. (5.10), we obtain

$$R(t) = e^{-\left(\frac{t}{\alpha(T_1)} + (\frac{1}{\alpha(T_0)} - \frac{1}{\alpha(T_1)}) \cdot t_1\right)^\beta}, \qquad t_1 \leq t < t_2 \tag{5.13}$$

More generally, the reliability function must satisfy the following continuity constraints:

$$R(t_\ell^-) = R(t_\ell^+), \qquad \ell = 1, 2, \cdots, m-1 \tag{5.14}$$

By generalizing the above calculation steps, the lifetime reliability of a processor at time t can be written as

$$R(t) = e^{-(\frac{t}{\alpha(T_\ell)} + \eta_\ell)^\beta}, \qquad t_\ell \leq t < t_{\ell+1} \tag{5.15}$$

where

$$\eta_\ell = \sum_{i=0}^{\ell-1} \left(\frac{1}{\alpha(T_i)} - \frac{1}{\alpha(T_{i+1})}\right) \cdot t_{i+1} \tag{5.16}$$

With Eq. (5.15)-(5.16), we can compute $MTTF$ by Eq. (5.3), but we still need to monitor the processor's temperature, which is obviously time-consuming.

Fortunately, since the tasks are executed periodically, the temperature variance with respect to time will be also periodical after it is stabilized. We hence can divide each period into the same subdivisions. Given each task execution period is divided into p time intervals, by Eq. (5.15)-(5.16), a processor's lifetime reliability at the end of first period is given by

$$R(t_p) = e^{-\left(\sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)}\right)^\beta} \tag{5.17}$$

Similarly, a processor's reliability at the end of the m^{th} period can be expressed as

$$R(t_{m \cdot p}) = e^{-\left(\sum_{i=0}^{m \cdot p-1} \frac{\Delta t_i}{\alpha(T_i)}\right)^\beta} \tag{5.18}$$

We notice that the changes of reliability function $R(t)$ in different periods are different; while $\sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)}$ does not vary from period to period. That is,

$$[-\ln R(t_{m \cdot p})]^{\frac{1}{\beta}} = \sum_{i=0}^{m \cdot p-1} \frac{\Delta t_i}{\alpha(T_i)} = m \sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)} = m \cdot [-\ln R(t_p)]^{\frac{1}{\beta}} \tag{5.19}$$

We therefore introduce the concept of *aging effect* of a processor in a period A, which enables us to integrate all lifetime reliability-related characteristics (including temperature, voltage, clock frequency, etc.) of a processor and its utilization together in this single value.

$$A = [-\ln R(t_p)]^{\frac{1}{\beta}} = \sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)} \tag{5.20}$$

Because typically $MTTF \gg t_p$, the $MTTF$ of a single processor defined as Eq. (5.3) can be approximated to

$$MTTF = \sum_{i=0}^{\infty} e^{-(i \cdot A)^{\beta}} \cdot t_p \tag{5.21}$$

The above is the lifetime estimation of a single processor. For an MPSoC platform, let us denote processor j's aging effect as A_j and its slope parameter as β_j and assume that there is no spare processors in the system (i.e., the system fails if one processor fails), the $MTTF$ of the entire system can be approximately expressed as

$$MTTF^{sys} = \sum_{i=0}^{\infty} e^{-\sum_{j}(i \cdot A_j)^{\beta_j}} \cdot t_p \tag{5.22}$$

While from the mathematical point of view the extension from the lifetime estimation of single processor to that of MPSoC platform is simply a production operation, Eq. (5.22) essentially does not lose any information about the correlation between processors. To clarify, let us consider an important feature of an MPSoC platform as an example, that is, the execution of a processor affects the temperature of its neighbors. When we estimate the system lifetime, the heating effect of processor j caused by not only itself but also other processors is reflected in its A_j. Similarly, the influence of processor j's behavior on others also affect their aging effect parameters. These A_j's, finally, bring the correlation between processors into the system lifetime estimation $MTTF^{sys}$.

5.5 Efficient MPSoC Lifetime Approximation

It is essential to be able to quickly evaluate the cost of a solution during the simulated annealing process because this task needs to be conducted whenever we find a solution. Calculating $MTTF^{sys}$ according to Eq. (5.22) directly, however,

is quite time-consuming, which limits our design space exploration capability. To tackle this problem, four speedup techniques are introduced in this section.

5.5.1 Speedup Technique I – Multiple Periods

Remind that the aging effect A_j of processor j is the same for every period. Obviously, its aging effect of ν periods can be expressed as $A_j \cdot \nu$. As long as the condition $t_p \cdot \nu \ll MTTF$ is still satisfied (i.e., ν is much less than the number of operational periods before permanent system failure), $MTTF^{sys}_{appxI}$ defined by the following equation can be a lifetime approximation.

$$MTTF^{sys}_{appxI} = \sum_{i=0}^{\infty} e^{-\sum_j (i \cdot A_j \cdot \nu)^{\beta_j}} \cdot t_p \cdot \nu \tag{5.23}$$

The idea behind this estimation is shown in Fig. 5.5, wherein the area inside the dotted curve is the system's actual MTTF while the approximated $MTTF^{sys}_{appxI}$ is the area inside the solid rectangles. As can be easily observed, although $MTTF^{sys}_{appxI}$ is not the accurate mean time to failure of the system, it is an effective indicator for the lifetime with different task schedules, because a task schedule with relatively larger MTTF tends to have larger $MTTF^{sys}_{appxI}$. This technique benefits us significantly in terms of computational time, i.e., ν times faster than the case without this technique.

5.5.2 Speedup Technique II – Steady Temperature

To obtain an accurate A_j used in Eq. (5.23) is a quite time-consuming process because the time interval $[t_i, t_{i+1})$ needs to be set as a very small value. Fortunately, the time for processors to reach steady temperature with task changes in the platform is typically much shorter than the execution time of tasks [28, 38]. As an example, we demonstrate the temperature variations for a sample MPSoC

platform containing three processors in Fig. 5.6(a), obtained from HotSpot [92], an efficient and accurate thermal simulator that is able to calculate transient and/or steady temperature of on-chip computing elements. Fig. 5.6(b) shows the corresponding processors that are under usage at a particular time. From this figure, we can observe that the processors stay at a relatively stable temperature most of the time when the tasks do not change. With this observation, we propose to calculate A_j at a much coarser time scale based on such steady temperature within each time slot as shown in Fig. 5.6(b).

5.5.3 Speedup Technique III – Temperature Pre-calculation

Even though A_j could be calculated efficiently with the above speedup techniques, we have to run HotSpot temperature simulator [92] to obtain the temperature information every time the simulated annealing algorithm reaches a solution. Let us perform a simple calculation. Suppose the initial and end temperature of algorithm is 10^2 and 10^{-5} respectively, cooling rate is 0.95, and 1000 neighbor solutions are

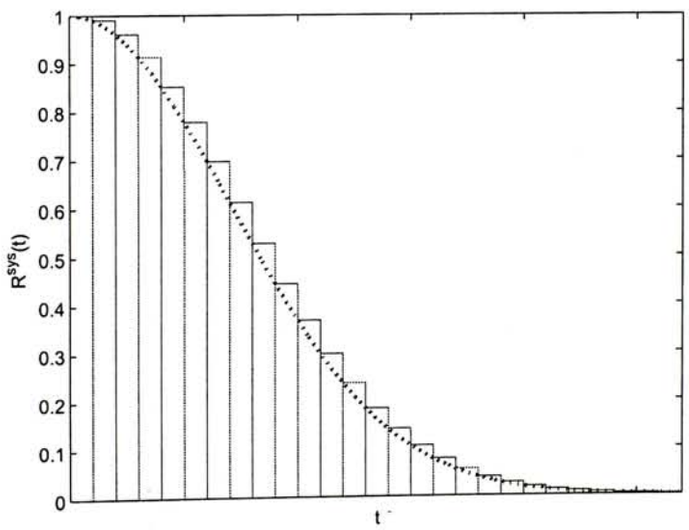
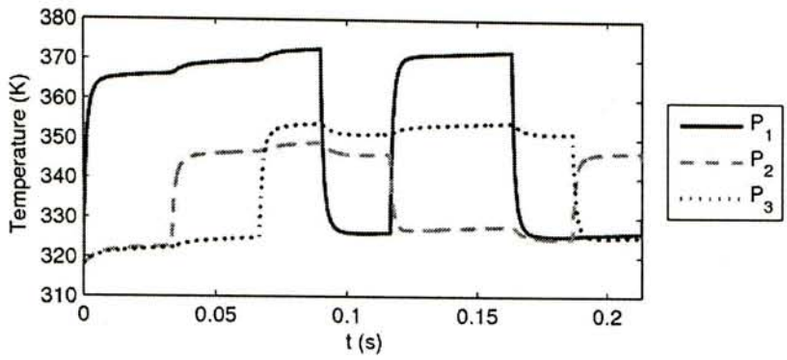
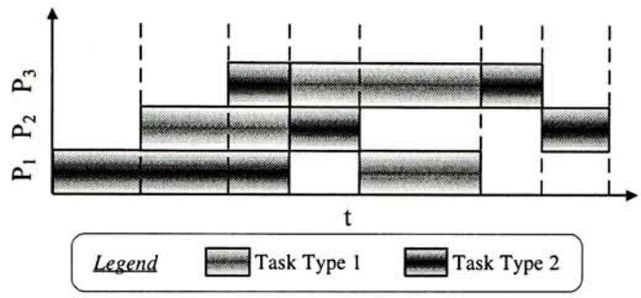


Figure 5.5: Approximation for the System's MTTF.



(a) Temperature Variation



(b) Slot Representation

Figure 5.6: An Example of Slot Representation and the Corresponding Temperature Variations.

searched at the same algorithm temperature, the time-consuming HotSpot simulator needs to be called $1000 \times \log_{0.95} \frac{10^2}{10^{-5}} \approx 3 \times 10^5$ times, which is obviously unaffordable. To avoid this problem, we propose to conduct the HotSpot simulation in a pre-calculation phase.

To pre-calculate the processors' temperatures, we define a series of *time slots* for task schedules. Each one is identified by the set of busy processors and the power consumption of the tasks running on these processors[1], as shown in Fig. 5.6. Since the power consumptions can be different when the same task execute on distinct processors and when different tasks execute on the same processor, the number of possible time slots is huge and it is very difficult, if not impossible, to

[1] In practice, the power consumption for a task may vary with different inputs, and hence we use the average power consumption here, as in [95].

run HotSpot once and pre-calculate the aging effect for all the cases. To tackle this problem, we categorize the tasks into m types (m is a user-defined value) based on their power consumptions when running on a processor and we assume the tasks belonging to the same category have the same power consumption value when they run on the same processor. Since every processor can be either used or unused in a time slot, and each processor in use has m possible power consumption values, there can be at most $\sum_{i=1}^{k} m^i \binom{k}{i} = \left((1+m)^k - 1\right)$ kinds of time slots in task schedules, where k is the quantity of processors on the platform. Here, m^i means that when i processors are in use, each has m possible power consumption values. In total, there are $\binom{k}{i}$ possible combinations that i-out-of-k processors are under-used. The possible values of occupied processor quantity i are $1, 2, \cdots, k$.

Denote by x_ℓ^i $(1 \leq i \leq k, 1 \leq \ell \leq m)$ the event that processor i is under usage in a time slot, and the task running on this processor belongs to type ℓ. Note that, the time slot here is independent of the length of time interval. We separate these two concepts based on the observation that as long as the time interval is not very short, its aging effect can be approximated by the steady temperature (as mentioned in Section 5.5.2), which is independent of the length. Each slot can be described by a set of x_ℓ^i, denoted by $\mathbf{X}$. We omit the processors in idle state in the representation of set $\mathbf{X}$. A task schedule is composed of a list of time slots. A feasible way to identify the time slot in a schedule is to cut the schedule into a series of time intervals at the task starting or ending time points. For example, suppose an embedded system contains 3 processors and its tasks are classified into 2 types, in the time order the schedule shown in Fig. 5.6(b) consists of 7 slots: $\{x_2^1\}, \{x_2^1, x_1^2\}, \{x_2^1, x_1^2, x_2^3\}, \{x_2^2, x_1^3\}, \{x_1^1, x_1^3\}, \{x_2^3\}, \{x_2^2\}$.

Let $T_j^{\mathbf{X}}$ be the steady temperature of processor j in time slot $\mathbf{X}$. Because the steady temperature depends on power consumption of processors and floorplan, and all are fixed in a slot, there are exactly $\left((1+m)^k - 1\right)$ possible $T_j^{\mathbf{X}}$ values for

processor j. Given the steady temperature of processor j in time slot $\mathbf{X}$ (i.e., $T_j^{\mathbf{X}}$), we calculate the aging effect factor of processor j, denoted as $\phi_j^{\mathbf{X}}$. Here *aging effect factor* is the aging effect in unit time, defined as

$$\phi_j^{\mathbf{X}} = \frac{1}{\alpha(T_j^{\mathbf{X}})} \tag{5.24}$$

For example, since $\mathbf{X}$ of the first slot in Fig. 5.6 is $\{x_2^1\}$, processor P_1's steady temperature is $T_1^{\{x_2^1\}}$. Its aging effect factor $\phi_1^{\{x_2^1\}}$ equals to $1/\alpha(T_1^{\{x_2^1\}})$. Given the length of the first slot Δt_0, P_1's aging effect in this slot is $\Delta t_0/\alpha(T_1^{\{x_2^1\}})$. It is necessary to highlight that, in any slot, not only under usage processors but also idle ones have aging effect. For processor P_1, we should also estimate its steady temperature and aging effect factor for the time slots where P_1 is not under usage (e.g., the 4th slots). The aging effect of P_1 in this schedule in a period can be computed by

$$\begin{aligned} A_1' = &\frac{\Delta t_0}{\alpha(T_1^{\{x_2^1\}})} + \frac{\Delta t_1}{\alpha(T_1^{\{x_2^1, x_1^2\}})} + \frac{\Delta t_2}{\alpha(T_1^{\{x_2^1, x_1^2, x_2^3\}})} + \frac{\Delta t_3}{\alpha(T_1^{\{x_2^2, x_1^3\}})} \\ &+ \frac{\Delta t_4}{\alpha(T_1^{\{x_1^1, x_1^3\}})} + \frac{\Delta t_5}{\alpha(T_1^{\{x_2^3\}})} + \frac{\Delta t_6}{\alpha(T_1^{\{x_2^2\}})} \end{aligned} \tag{5.25}$$

The aging effect of other processors in a period can be computed in the same method.

Then, combining the speedup techniques II and III, for a task schedule we can compute A_j' for every processor j. Replacing the accurate A_j in Eq. (5.23) with A_j' yields

$$MTTF_{appxII}^{sys} = \sum_{i=0}^{\infty} e^{-\sum_j (i \cdot A_j' \cdot \nu)^{\beta_j}} \cdot t_p \cdot \nu \tag{5.26}$$

5.5.4 Speedup Technique IV – Time Slot Quantity Control

We notice that the number of possible time slots $\left((1+m)^k - 1\right)$ increase exponentially with the increase of on-chip processor cores m. This issue can be effectively resolved based on the observation that when a core is in execution, usually only nearby cores' temperatures are affected. Therefore, we can identify those neighboring processor cores based on the MPSoC's floorplan and pre-calculate the temperatures for a much less number of time slots. In practice, the processor cores on an MPSoC platform oftentimes do not crowd together (i.e., separated by other functional blocks), and hence can be naturally divided into a few regions and we conduct temperature estimation for them separately during the pre-calculation phase.

5.6 Experimental Results

5.6.1 Experimental Setup

To evaluate the effectiveness and efficiency of the proposed methodology, we conduct experiments on a set of random task graphs generated by TGFF [32] running on various hypothetical MPSoC platforms. The number of tasks ranges from 20 to 260, and the maximum in- and out-degree of a task is set to be the default values used in TGFF (i.e., 3 and 2, respectively). The number of processor cores varies between 2 and 8. By the speedup technique IV, a large platform that contains 6 or 8 processors is partitioned into two domains for pre-calculation. Unless specified otherwise, all the speedup techniques presented in Section 5.5 are applied on the proposed algorithm for approximation. We have also considered the homogeneity of platforms. For homogeneous platforms, all processor cores have the same execution time for a certain task. For heterogeneous ones, two kinds of processor cores are assumed: main processors and co-processors. The former ones have relatively

higher processing capability than the latter ones in most cases. For all task graphs and platforms, we compare the proposed strategy with an existing thermal-aware task allocation and scheduling algorithm proposed in [111] (abbreviated in tables to thermal-aware). List scheduling is utilized in [111], i.e., a list of unscheduled tasks is maintained and the task with the highest priority is scheduled iteratively in a deterministic manner. To reduce the peak temperature, task energy consumptions are taken into consideration in [111] when calculating the priority. Once the task schedule is constructed, its makespan (i.e., the time interval that all periodical tasks need to finish their executions once) becomes known. For fair comparison, it is used as the reference deadline for the proposed approach.

The simulated annealing parameters are set as follows: initial temperature = 100, cooling rate = 0.95, end temperature = 10^{-5}, and the number of random moves at each temperature = 1000. Moreover, because of the lack of public empirical data on the weight of influence of various failure mechanisms on real circuit, we use the electromigration failure model presented in [39] in our experiments[2]. The corresponding parameters are set as the cross-sectional area of conductor $A_c = 6.4 \times 10^{-8} cm^2$, the current density $J = 1.5 \times 10^6 A/cm^2$ and the activation energy $E_a = 0.48eV$. Further, the power density of platforms is in the range of 3.33 to 12.5 W/cm^2; and the tasks are categorized into 3 groups depending on their power consumption. The slope parameter in Weibull distribution used for describing the processor cores' lifetime reliability in homogeneous platforms is set as $\beta = 2$. While in heterogeneous ones, the slope parameters of main processors and co-processors are set to 2.5 and 2, respectively. Unless otherwise specified, the clock frequency of the main processors in heterogeneous platforms is set to be twice of that of the co-processors and the one in homogeneous platforms, i.e., the *frequency ratio* is two.

[2] Our model can be applied to other failure mechanisms as well. We can also combine the effect of multiple failure mechanisms and derive an overall MTTF based on [95, 97].

In addition, we define a reference platform, which contains a single processor core with a fixed temperature $351.5K$, slope parameter of Weibull distribution $\beta = 2$, and the same clock frequency as the processor cores in homogeneous ones. Its $MTTF$ is set to be 1000 units. The $MTTF$ obtained in our experiments are normalized to this reference case for easier comparison.

5.6.2 Results and Discussion

Let us first validate the approximation techniques used for MTTF estimation. By using our algorithm, we obtain a set of valid task schedules (i.e., the task schedules that meet the deadlines) for a homogeneous 2-processor platform. For each schedule, the approximated $MTTF$ are computed using Eq. (5.26), where ν is set to 100. Then, we derive the accurate $MTTF$ values by monitoring the temperature variation using HotSpot for the same schedules, and compare them to the approximated values. As shown in Fig. 5.7, our approximation is able to reflect the quality of different valid schedules. That is, if a schedule has larger mean time to failure, it tends to have larger approximated value. Also, it is worth noting that because of exponentially increased CPU execution time overhead with respect to the number of processors in the platform, we were not able to provide accurate $MTTF$ for larger platforms[3].

Next, we present experimental results obtained with various platforms and task graphs in Table 5.2. The detailed description of test cases are listed in Table 5.1, where Column 2-3 describe the task graph; Column 4-5 indicate the number of main processors and co-processors on the platform; Column 6 is the makespan obtained by thermal-aware task allocation and scheduling algorithm in [111] and is used as the baseline deadline of our algorithm.

As shown in Table 5.2, in most cases the results obtained with our algorithm

[3]The MTTF values shown in the following experiments are approximated ones.

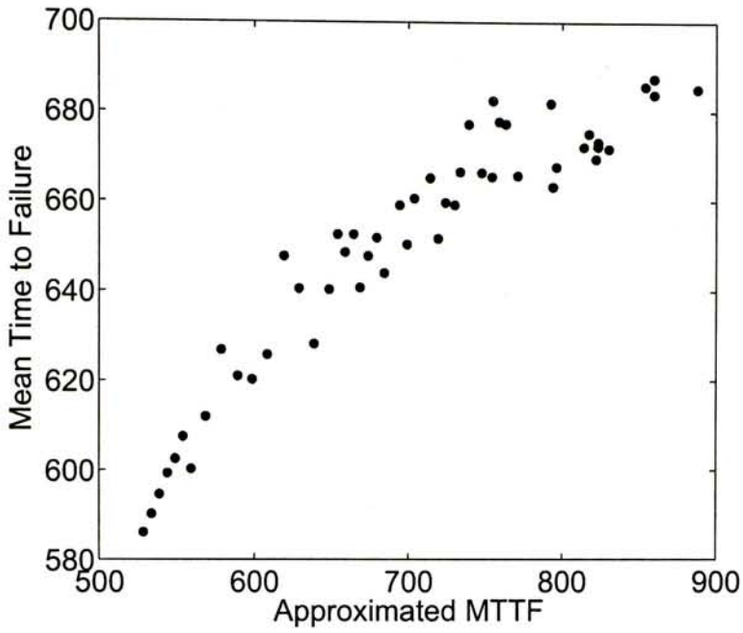


Figure 5.7: Comparison between Approximated $MTTF$ and Accurate Value.

Test Case Index	Task Description		Platform Description		Deadline (s)
	Task	Edge	Main PE	Co-PE	
$\mathcal{T}_1$	22	23	2	0	535
$\mathcal{T}_2$	49	76	4	0	1106
$\mathcal{T}_3$			2	2	697
$\mathcal{T}_4$	76	106	6	0	918
$\mathcal{T}_5$			2	4	676
$\mathcal{T}_6$	131	190	8	0	1227
$\mathcal{T}_7$			2	6	984

Table 5.1: Test Cases.

have longer lifetime than that of thermal-aware one even if the deadlines of both algorithms are the same (see Column 3-4). The only exception is the "2 processors 22 task" case (Row 4). When the same deadline is assumed, we observe the same lifetime resulted from our algorithm and that in [111]. We attribute this phenomenon to the simple MPSoC and the small task graph. That is, in this case, the schedules that are able to meet the deadlines are quite limited and we are not able

Test Case Index	Thermal-Aware [111]	Simulated Annealing					
		0% $\mathcal{DR}$		5% $\mathcal{DR}$		10% $\mathcal{DR}$	
	$MTTF$	$MTTF$	Δ (%)	$MTTF$	Δ (%)	$MTTF$	Δ (%)
$\mathcal{T}_1$	492.47	492.47	0	582.30	18.24	582.30	18.24
$\mathcal{T}_2$	216.05	226.87	5.01	247.31	14.47	263.38	21.91
$\mathcal{T}_3$	137.44	161.33	17.38	171.20	24.56	185.59	35.03
$\mathcal{T}_4$	228.87	239.91	4.82	256.73	12.17	273.28	19.40
$\mathcal{T}_5$	97.18	125.07	28.70	137.93	41.93	150.00	54.35
$\mathcal{T}_6$	227.24	235.78	3.76	250.86	10.39	265.56	16.86
$\mathcal{T}_7$	88.00	121.78	38.09	136.38	54.64	155.50	76.31

Δ: Difference ratio between $MTTF$ of SA and that of thermal-aware;
$\mathcal{DR}$: Deadline relaxation;

Table 5.2: Lifetime Reliability of Various MPSoC Platforms with Different Task Graphs.

Task Description	Thermal-Aware [111]	Simulated Annealing		
		$\mathcal{DR}$ (%)	$MTTF$	Δ (%)
# of Task: 101 # of Edge: 142	Deadline: 1059s $MTTF$:240.07	0	247.79	3.21
		5	264.25	10.07
		10	279.64	16.48
# of Task: 131 # of Edge: 190	Deadline: 1227s $MTTF$:227.24	0	235.78	3.76
		5	250.86	10.39
		10	265.56	16.86
# of Task: 201 # of Edge: 292	Deadline: 1809s $MTTF$:207.26	0	221.03	6.64
		5	235.95	13.84
		10	250.00	20.62
# of Task: 251 # of Edge: 366	Deadline: 2014s $MTTF$:191.38	0	203.37	6.27
		5	216.56	13.16
		10	230.17	20.27

Table 5.3: Lifetime Reliability of 8-Processor Homogeneous Platforms.

to find a solution with extended MTTF. If we relax the deadline by 5% or 10%, the advantage of the proposed lifetime reliability-aware task scheduling algorithm is more obvious (see Column 5-8). Taking the last row as an example, with the deadline relaxation, the lifetime extension ratio increases from 38.09% to 54.64%, and to 76.31%. Also, we notice that our algorithm provides more benefit if the

Task Description	Thermal-Aware [111]	Simulated Annealing		
		$\mathcal{DR}$ (%)	$MTTF$	Δ (%)
# of Task: 101 # of Edge: 142	Deadline: 809s $MTTF$:91.64	0	129.04	40.81
		5	146.01	59.33
		10	160.50	75.14
# of Task: 131 # of Edge: 190	Deadline: 984s $MTTF$:88.00	0	121.78	38.09
		5	136.38	54.64
		10	155.50	76.31
# of Task: 201 # of Edge: 292	Deadline: 1416s $MTTF$:85.27	0	130.42	52.95
		5	143.71	68.54
		10	149.71	75.57
# of Task: 251 # of Edge: 366	Deadline: 1693s $MTTF$:85.73	0	124.21	44.89
		5	137.88	60.83
		10	151.10	76.25

Table 5.4: Lifetime Reliability of 8-Processor Heterogeneous Platforms.

platform is a heterogenous one. For example, when we relax the deadline by 5%, the lifetime improvement on heterogeneous 6-processor platform is 41.93%, much higher than that on homogenous 6-processor platform, which is 12.17%. This is mainly because, for heterogeneous platforms, the thermal-aware task allocation and scheduling algorithm in [111] is based on list scheduling technique and tends to assign tasks to main processors because the main processors have better performance. In this case it is very likely that the aging effect of the main processors were much serious than that of the co-processors, while our algorithm is able to achieve more balanced aging among these processors.

A closer observation for 8-processor platforms is shown in Table 5.3 and Table 5.4. For the same platform, more task graphs are scheduled on it. When we target a larger task graph, the lifetime improvement obtained by our algorithm tends to be larger. For example, if we relax the deadline constraints by 10%, the lifetime improvements on the homogeneous platform for a task graph with 131 tasks and that with 201 tasks are 16.86% and 20.62%, respectively. We attribute it to the

more valid solutions with larger number of tasks. However, it should be noted that the effectiveness of the proposed methodology also depends on many other factors, such as the detailed precedence dependencies.

Fig. 5.8 shows the difference ratio between the MTTF obtained from the proposed approach and [111], as a function of frequency ratio. Here, frequency ratio γ is an important factor that reflects platform heterogeneity, representing that the clock frequency of the main processors is set to be $\gamma\times$ of that of the co-processors. We schedule the task graph with 131 tasks on the 8-core heterogeneous platform. The benefit provided by the proposed approach significantly increases as the ratio between main processors and co-processors grows. In other words, the proposed method performs better when the heterogeneity of the MPSoC system is high. For example, as frequency ratio increases from 1 to 4, even when no deadline relaxation is allowed, the lifetime extension ratio grows by a factor of 2.65. Also, the improvement achieved by deadline relaxation is more significant when the system heterogeneity is high. Consider the deadline extension by 5% as an example. The lifetime extension ratio improvements are 7.33% and 20.47% for the same frequency ($1\times$) case and $4\times$ case, respectively.

We are also interested in the trade-off between performance and mean time to failure. The experimental results for two sample cases, a heterogeneous 8-processor platform and a homogeneous 4-processor one, are shown in Fig. 5.9. We can observe that the $MTTF$ generally increases with the relaxation of deadlines. This is mainly because the flexibility of selecting task schedules increases with respect to the deadline relaxation. We can also observe that when the deadline constraint is relaxed to a certain point (e.g., deadline relaxation exceeds 160% in Fig. 5.9(b)), $MTTF$ starts to saturate. This is because the task schedule with the longest lifetime does not violate deadline constraints and has been selected as the solution.

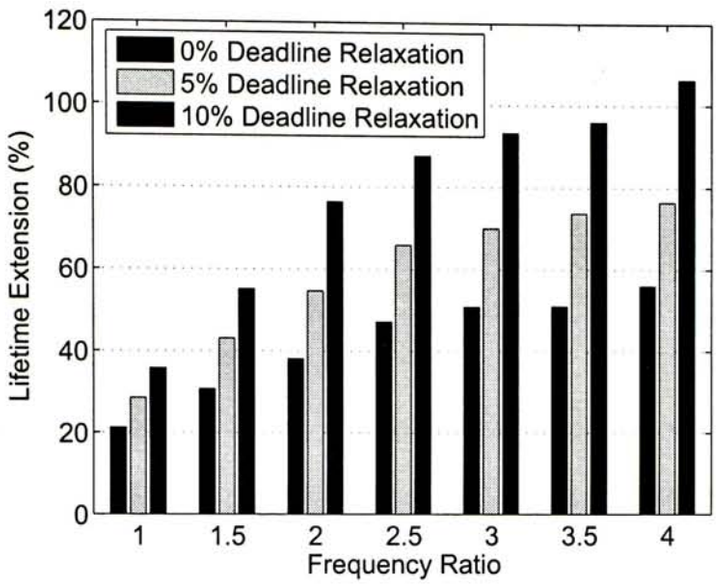
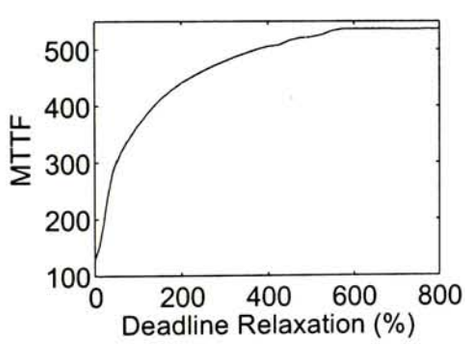
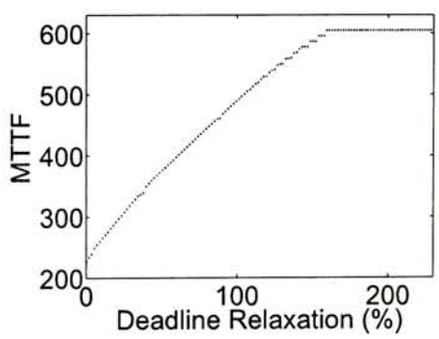


Figure 5.8: The Impact of Heterogeneity on the Effectiveness of the Proposed Strategy.



(a) Heterogeneous 8-Processor Platform



(b) Homogeneous 4-Processor Platform

Figure 5.9: The Extension of $MTTF$ with the Relaxation of Deadlines.

Finally, as for the efficiency of our algorithm, the simulated annealing process requests 50–200s of CPU time on Intel(R) Core(TM)2 CPU 2.13GHz for each case in our experiments. For example, "4 processors 49 tasks" needs 84 seconds, while "8 processors 101 tasks" costs 158 seconds. The CPU time spending on pre-calculation (i.e., steady temperature estimation of time slots) ranges from 3 to 160

seconds. We have tried the pre-calculation for 8-processor platform without partitioning the platform into two regions. As expected, it requires extremely long CPU time (more than 5 hours), which illuminates the need of time slot quantity control (speedup technique IV). We also attempted to classify the tasks into 5 groups and keep the platform partitioning, the pre-calculation for 8-processor platform needs around 12 minutes. The impact on MTTF accuracy highly depends on the floorplan of MPSoC platforms. In particular, when the processor cores are placed on the silicon die in a loose manner, we observe negligible temperature difference between the cases with and without time slot quantity control. In contrast, if the cores are centered in a small region, the temperature difference could be a few centigrade.

5.7 Conclusion and Future Work

The lifetime reliability of MPSoC designs has become a serious concern for the industry with technology scaling. To tackle this problem, different from prior work, we propose a simulated annealing-based task allocation and scheduling strategy that maximizes the lifetime of platform-based MPSoC designs under performance constraints. In order to efficiently estimate the lifetime reliability of various solutions with acceptable accuracy, we propose an analytical model and several speedup techniques, by explicitly taking the aging effect of processors into account. Experimental results show that the proposed techniques significantly extend the lifetime of platform-based MPSoC designs, especially for heterogeneous platform with large task graphs.

While our current solution has considered the possible processor cores' voltage/frequency differences and hence is applicable for designs with multiple voltage-frequency islands, we assume they do not change at runtime. Since DVFS has been employed in many MPSoC platforms, we plan to take this effect into account in

our future work.

□ **End of chapter.**

Chapter 6

Energy-Efficient Task Allocation and Scheduling

Part of content in this chapter is included in the proceedings of *IEEE/ACM Design, Automation, and Test in Europe (DATE)* 2010 [49].

6.1 Introduction

With the ever advancement of semiconductor technology, designers are now able to integrate several microprocessors, dedicated digital hardware and mixed-signal circuits on a single silicon die, known as MPSoC. In response to today's uncertain electronics market, when designing complex embedded systems, it is increasingly popular to employ pre-designed MPSoC platforms and map applications onto them to reduce design risk and achieve short time-to-market [83]. Various platforms with specific functionalities reflecting the need of the expected application domain have been developed in the industry recently, e.g., ARM PrimeXsys platform [7], IBM/Sony/Toshiba Cell platform [93], and NXP Nexperia platform [81].

When building platform-based embedded systems, one of the basic issues is

to conduct task allocation and scheduling for applications, in which the allocation of tasks is to effectively utilize the available processors while the scheduling is to meet various constraints (e.g., timing constraints for real-time tasks). Recently, minimizing energy consumption has become a critical task for embedded system designs, especially in battery-powered electronic products. A widely-used technique for energy reduction is dynamic voltage scaling (DVS) [58], by which we scale down the voltage/frequency of individual processors according to temporal performance requirements of applications. Various energy-efficient task allocation and scheduling techniques for DVS-enabled embedded systems have been presented in the literature(e.g., [9, 27]).

Despite the significant advancement of platform-based embedded system design methodologies in recent years, only limited works have considered the lifetime reliability of the system [52, 118]. With the ever-increasing on-chip power and temperature densities, however, the wear-out failures (e.g., electromigration on the interconnects and time dependent dielectric breakdown in the gate oxides) have become serious concerns for the industry [16, 64, 95]. As shown in [78, 94], the failure rates for electronic products can be quite high within its warrantee period and the main reason for such high failure rate was traced to excessive stress on the embedded processors.

Existing energy-efficient task allocation and scheduling techniques target at reducing the overall energy consumption of the system. The lifetime of the system, however, is determined by the component with the shortest service life. Consequently, it is likely that the solution with the minimum energy consumption result in excessive stress on certain components, leading to unexpected low lifetime reliability of the system. In addition, today's complex embedded systems usually do not stick to a single execution mode throughout their entire service lives. Instead, they work across a set of different interacting applications and operational modes.

For instance, modern smart phones not only provide communication service, but also work as MP3 player, video decoder, game console, and digital camera. For such *multi-mode embedded systems* [79], the above problem can be further exacerbated due to inter-mode resource sharing and the associated possible imbalanced usage of processor cores. For example, an energy-efficient processor in a heterogeneous MPSoC platform might be much more heavily used in every operational mode, thus aging much faster than other embedded processor due to the excessive stress on it.

From the above, *it is essential to explicitly consider lifetime reliability issue in energy-efficient embedded system designs*. To tackle this problem, in this paper, we first show how to conduct energy-efficient task allocation and scheduling on MPSoC platforms for a single execution mode of the embedded system, under the consideration of the lifetime reliability constraint of the system. For multi-mode embedded systems, since the overall system's lifetime reliability is also related to the mode execution probabilities, it is not necessary to apply the same constraint to every execution mode. That is, we can afford to have task allocation and schedules for certain modes with lower reliability if such solutions reduce energy consumption dramatically, and compensate the reliability loss from other execution modes. Based on this observation, we propose to identify a set of "good" task allocation and schedule solutions in terms of lifetime reliability and/or energy consumption for each execution mode. Then, we introduce novel methodologies to obtain an optimal combination of task schedules that minimizes the energy consumption of the entire multi-mode system, while satisfying given systemwide lifetime reliability constraint. Experimental results on various hypothetic multi-mode MPSoC platforms show that the proposed solution is able to significantly reduce the system energy consumption under reliability constraint.

The reminder of this paper is organized as follows. Section 6.2 presents pre-

liminaries of this work and formulates the problem studied in this paper. The analytical models used to estimate performance, energy consumption and lifetime reliability of embedded systems are presented in Section 6.3. In Section 6.4, we present our task allocation and scheduling algorithm for single-mode MPSoC embedded systems. Then, we extend it for multi-mode embedded systems in Section 6.5. Section 6.6 presents our experimental results for hypothetical MPSoC platforms. Finally, Section 6.7 concludes this work.

6.2 Preliminaries and Problem Formulation

6.2.1 Related Work

Recently, a major trend in embedded system designs is towards energy-efficient computing based on the concept of performance on demand. That is, most embedded systems are characterized by a time-varying computational load and significant energy savings can be achieved by recognizing that peak performance is not always required and hence the operational voltage and frequency of the processors can be dynamically adjusted based on instantaneous processing requirement. The energy-efficient task allocation and scheduling problem for MPSoCs can thus be formulated as: Given well-characterized task graphs representing a functional abstraction of the applications running on a pre-selected MPSoC platform, designers are to allocate tasks to different processors, schedule these tasks and assign voltage/frequency for processor cores in distinct operational periods to minimize the energy consumption of the system, under the consideration of various constraints (e.g., meeting deadlines for real-time tasks).

A number of energy-efficient design methodologies have been presented in the literature to tackle the above problem (e.g., [9, 27]). These techniques mainly resort to DVS and slack reclaiming to cut down the energy consumption of the em-

bedded processors. In particular, Schmitz *et al.* [85] proposed an energy-efficient co-synthesis framework for multi-mode embedded systems under the consideration of mode execution probabilities, in which a single execution mode occupies the entire MPSoC at a time.

At the same time, with aggressive technology scaling, the lifetime reliability of today's high-performance integrated circuits has also become a major concern for the industry [16]. The *wear-out failure mechanisms* that lead to permanent errors of IC products include electromigration on the interconnects, TDDB in the gate oxides, NBTI on PMOS transistors, and thermal cycling, and they were shown to be highly related to the temperature and voltage applied to the circuit. Thus, existing thermal-aware task scheduling techniques may improve the MPSoC's lifetime reliability implicitly, by balancing different processors' temperatures or keeping them under a safe threshold. However, as pointed out in [52], since circuit wear-out failures are also dependent on many other factors (e.g., internal structure, operational frequency and voltage), without *explicitly* taking the lifetime reliability into account during task allocation and scheduling, various processor cores may still age differently and thus result in shorter lifetime for the MPSoC-based embedded system.

Recently, the authors of [52] proposed a novel analytical model for the lifetime reliability of multiprocessor platforms, which, unlike prior work (e.g., [29, 95]), is able to capture the processors' accumulated aging effect. They have also introduced a simulated annealing technique to maximize the service life of MPSoC-based embedded systems under performance constraint. However, energy consumption issues are not considered in [52] and it focused on single execution mode only. In practice, it is not necessary to prolong the service life of embedded systems as much as possible. Rather, energy minimization should be the primary optimization objective with given lifetime reliability being used as a constraint.

6.2.2 Problem Formulation

Based on the above, the problem investigated in this paper is formulated as follows:

Problem: Given

- the floorplan of the platform-based MPSoC embedded system that consists of ℓ processor cores;
- n execution modes. Each mode i is represented by a directed acyclic task graph $\mathbf{G}_i = (\mathbf{V}_i, \mathbf{E}_i)$, wherein each node in $\mathbf{V}_i$ indicates a task in $\mathbf{G}_i$, and $\mathbf{E}_i$ is the set of directed arcs that represent precedence constraints;
- the joint probability density function[1] that the system is in various modes $f_{Y_1,Y_2,\cdots,Y_n}(y_1, y_2, \cdots, y_n)$, where y_i represents the probability that the system is in execution mode i;
- the execution time $w_{i,j,k}$ of task j of mode i on processor k under maximum supply voltage V_{dd};
- the power consumption $P_{i,j,k}$ of task j of mode i on processor k under maximum supply voltage V_{dd};
- deadline $d_{i,j}$ of task j of mode i, meaning that task j in $\mathbf{G}_i$ should be finished before $d_{i,j}$;
- target service life L and the corresponding reliability requirement η;
- failure mechanism parameters (e.g., activation energy E_a of electromigration) and the corresponding failure distributions;

[1]The mode execution probabilities can be estimated based on the methods shown in [85].

Determine a periodical task allocation and schedule on the given MPSoC platform for each execution mode such that the expected energy consumption is minimized, under the performance constraints that real-time tasks are finished before deadlines and the lifetime reliability constraint that the system reliability at the target usage life is no less than η.

Note that, for the ease of discussion, in this work, we assume each execution mode in a multi-mode system corresponds to one directed acyclic task graph only. Our proposed approach, however, could be easily extended to handle multiple task graphs for a single mode by constructing a hyper task graph and performing task scheduling on a hyper-period [65], if necessary. In addition, while there are other hardware resources in the MPSoC platform consuming energy and suffering from wear-out failures, we mainly consider processor cores in this work due to their heavy stress and varying operational behaviors. Our work can be easily extended to take other hardware components (with simpler activities) into account, if needed.

6.3 Analytical Models

With the above given problem formulation, this section describes the analytical models used in this work to calculate the performance, the energy consumption [59, 74] and the lifetime reliability [52] for task allocation and schedule solutions in DVS-enabled systems.

6.3.1 Performance and Energy Models for DVS-Enabled Processors

The power dissipation of a processor core can be described by the following equation [59]

$$\begin{aligned} P &= P_{on} + P_{dynamic} + P_{leakage} \\ &= P_{on} + C_{eff} \cdot V_{dd}^2 \cdot f + (V_{dd} \cdot I_{subn} + |V_{bs}| \cdot I_j) \end{aligned} \tag{6.1}$$

where, P_{on} is the inherent power cost for keeping the processor on, which can be assumed as a constant value [59]; $P_{dynamic}$ represents the dynamic power consumption, which is quadratic dependent on supply voltage V_{dd} and proportional to operating frequency f and the effective switching capacitance C_{eff}; $P_{leakage}$ indicates the leakage power dissipation due to reverse bias junction current I_j and subthreshold leakage current I_{subn}. V_{bs} is the body bias voltage; I_j can be approximated as a constant value, as pointed out in [74]; I_{subn} is a voltage-dependent parameter, given by

$$I_{subn} = K_1 e^{K_2 V_{dd}} \tag{6.2}$$

where, K_1 and K_2 are constant fitting parameters.

The operational frequency of a processor core is bounded by its slowest path, whose delay can be expressed in terms of supply voltage V_{dd}, threshold voltage V_{th}, and logic depth of the critical path L_d [74], that is,

$$\tau = \frac{L_d K_3}{(V_{dd} - V_{th})^{\kappa}} \tag{6.3}$$

where, κ is a material-related constant. K_3 is a constant related to process technology. As the operational frequency is inversely proportional to circuit delay, we have

$$f = \frac{1}{\tau} \tag{6.4}$$

We define the voltage scaling parameter ρ as a value within the range $[0, 1]$, where $\rho = 1$ implies the processor is working under maximum supply voltage.

The scaled supply voltage is therefore ρV_{dd}. Substituting it into Eq. (6.3) and Eq. (6.4) yields the scaling operating frequency, i.e.,

$$f(\rho) = \frac{(\rho V_{dd} - V_{th})^{\kappa}}{L_d K_3} \tag{6.5}$$

We also redefine the power consumption of a task (that is, Eq. (6.1)) with voltage scaling parameter ρ as

$$P(\rho) = C_{eff} \cdot \rho^2 V_{dd}^2 \cdot f(\rho) + \rho V_{dd} \cdot K_1 e^{K_2 \cdot \rho V_{dd}} + |V_{bs}| \cdot I_j + P_{on} \tag{6.6}$$

Note that, without voltage scaling (i.e., $\rho = 1$) the execution time of this task is simply w, that is, $w(1) \equiv w$. By this normalization, we can express the execution time given voltage scaling parameter ρ by

$$w(\rho) = \frac{(V_{dd} - V_{th})^{\kappa}}{(\rho V_{dd} - V_{th})^{\kappa}} \cdot w \tag{6.7}$$

Similarly, the power consumption can be also normalized by calculating the effective switching capacitance C_{eff} with $P(1) \equiv P$. The material and technology-related parameters in this model (e.g., K_1) are set according to [74] with $70nm$ technology.

Given $d_{i,j}$ the deadline for each task j in the task graph for execution mode i, the deadline of entire task graph is $\max_j\{d_{i,j}\}$. With Eq. (6.6)-(6.7), we divide the total energy consumption of all tasks in a period by the schedule deadline for execution mode i, yielding

$$\begin{aligned} E_i &= \frac{1}{\max_j\{d_{i,j}\}} \cdot E_{i,period} \\ &= \frac{1}{\max_j\{d_{i,j}\}} \sum_j \sum_k P_{i,j,k}(\rho_{i,j,k}) \cdot w_{i,j,k}(\rho_{i,j,k}) \cdot 1_{\{i,j \to k\}} \end{aligned} \tag{6.8}$$

where, $1_{\{i,j \to k\}}$ is an indicator function, representing that task j is assigned to processor k. If so, it equals 1; otherwise, it is 0.

The above model can be used to evaluate the energy consumption of task schedules for a single execution mode. To express the energy consumption of multi-mode system, we need to consider the stress weight of various execution modes. Given y_i the probability of mode i's execution, we have

$$\begin{aligned} E &= \sum_i y_i \cdot E_i \\ &= \sum_i \frac{y_i}{\max_j\{d_{i,j}\}} \sum_j \sum_k P_{i,j,k}(\rho_{i,j,k}) \cdot w_{i,j,k}(\rho_{i,j,k}) \cdot 1_{\{i,j \to k\}} \end{aligned} \tag{6.9}$$

6.3.2 Lifetime Reliability Model

The calculation of the lifetime reliability for embedded processors is not an easy task as it depends on some time-varying parameters (e.g., temperature). Because of this, most prior work assumes exponential failure distribution due to its mathematical tractability (e.g., [29, 95]). This assumption, however, cannot capture the accumulated aging effect of IC hard errors. To resolve this problem, in this work, we resort to the *aging effect* concept proposed in [52], in which general failure distributions (e.g., Weibull distribution with increasing failure rate [1]) are considered. This *aging effect*, being used to capture the stress of a task schedule on a processor in one period, has a nice property that it does not vary among periods. Because of this, we are able to obtain the system's lifetime reliability at any arbitrary time by collecting the aging-related parameters for one period only instead of the entire service life.

Let us use $A_{i,k}$ to denote the aging effect of processor k in a period. Suppose the system remains in mode i through its service life, we can express system reliability that follows Weibull distribution at the end of target service life L as

$$R_i = \exp\left[-\sum_k \left(\frac{A_{i,k}}{\max_j\{d_{i,j}\}} \cdot L\right)^{\beta_k}\right] \tag{6.10}$$

where, β_k is the slope parameter of processor k in its Weibull failure distribution.

To reduce computational complexity, instead of monitoring the fine-grained temperature variation using temperature simulator (e.g., HotSpot [92]), given the power consumption of a task j running on processor k in execution mode i, we calculate its steady temperature as follows.

$$T^{ss}_{i,j,k} = P_{i,j,k}(\rho_{i,j,k}) \cdot C_k + T_{amb} \tag{6.11}$$

where, C_k is the thermal capacitance, and T_{amb} is the ambient temperature. The accuracy of such approximation is acceptable because the time for processor cores to reach steady temperature is typically much shorter than the task execution time [28, 52].

Then, we express the aging effect caused by task j on process k as the product of the scaled execution time $w_{i,j,k}(\rho_{i,j,k})$ and the aging effect in unit time $\alpha_k(T^{ss}_{i,j,k})$ provided temperature $T^{ss}_{i,j,k}$, i.e.,

$$A_{i,j,k} = w_{i,j,k}(\rho_{i,j,k}) \cdot \alpha_k(T^{ss}_{i,j,k}) \tag{6.12}$$

where, the function $\alpha_k(T)$ is defined by existing failure models or the combined effect of multiple failure mechanisms [52].

With the additivity of aging effect caused by various tasks (as proved in [52]), the aging effect of a task schedule is the summation of that caused by every task in a period, that is,

$$A_{i,k} = \sum_j A_{i,j,k} \cdot 1_{\{i,j \to k\}} + \frac{d_i - \sum_j w_{i,j,k}(\rho_{i,j,k}) \cdot 1_{\{i,j \to k\}}}{\alpha_k(T_{amb})} \tag{6.13}$$

It is important to note that, we also consider the aging effect in idle state (the second term).

Again, this mode is based on the assumption that the system remains in mode i throughout its lifetime. Therefore, we use it to estimate the task schedules for a single mode. With the aging effect additivity, we compute the reliability constraint for multi-mode system by

$$R = \exp\left[-\sum_{k}\left(\sum_{i}\frac{y_i \cdot A_{i,k}}{\max_j\{d_{i,j}\}} \cdot L\right)^{\beta_k}\right] \geq \eta \tag{6.14}$$

6.4 Proposed Algorithm for Single-Mode Embedded Systems

In this section, we study energy-efficient task allocation and scheduling for single-mode embedded systems, in which we try to minimize energy consumption under both performance and reliability constraints, using simulated annealing-based techniques.

Generally speaking, DVS is helpful to both energy savings and reliability improvement, as shown in Fig. 6.1. Therefore, we propose to solve the energy-efficient task allocation and scheduling problem in two phases. In the first phase, we try to obtain optimized task allocation and schedules without considering DVS; while in the second phase, we make use of the timing slacks to determine appropriate voltage scales for DVS-enabled processors to minimize energy under performance and lifetime reliability constraint.

6.4.1 Task Allocation and Scheduling

The most straightforward representation for task allocation and schedule solutions is (*schedule order sequence*; *resource binding sequence*), which can be used to construct task schedules directly and corresponds to unique reliability, energy consumption and schedule length (also known as *makespan*). This simple represen-

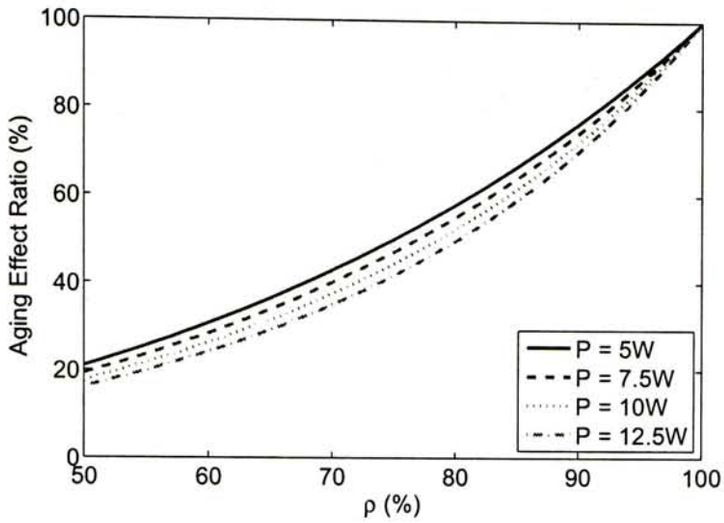


Figure 6.1: Influence of Voltage Scaling on Aging Effect.

tation, however, leads to huge design exploration space. Let v_i be the number of nodes in the task graph for execution mode i. With ℓ processors in the system, the solution exploration space is as high as $(v_i! \cdot \ell^{v_i})$.

Fortunately, we notice that, without considering DVS, both energy consumption and lifetime reliability depend mainly on task allocation and are almost independent of their schedule orders (see Eq. (6.8) and Eq. (6.10)). In addition, with known task allocation, there is a rich literature (e.g., [68, 69]) on how to conduct task scheduling to meet real-time task deadlines and reduce the total schedule length. Based on the above, we propose to represent the solutions with task allocation only. For example, for a three-processor system, (P_1, P_2, P_1, P_3, P_3) means that task 1, 2, 3, 4, 5 are assigned to processor 1, 2, 1, 3, 3, respectively. With this representation, the solution space in our simulated annealing process shrinks to ℓ^{v_i}. The random move strategy is also quite simple with this representation, that is, every time we randomly pick up a task and modify its resource binding (i.e., assign this task to another processor core, different from the original one).

As the simulated annealing searching procedure is guided to the solution with

minimum cost, we should define a cost function such that the task schedule with higher reliability and lower energy consumption has less cost. Also, to meet the performance constraint, a heavy penalty should be given to the task assignment that cannot meet this constraint. Denoting by $m_{i,j}$ the earliest finish time of task j given resource binding of mode i, we therefore define the cost function as

$$C_i = \gamma \cdot 1_{\{\exists j:\, m_{i,j} > d_{i,j}\}} + \widetilde{R}_i \times E_i \tag{6.15}$$

Here, γ represents a significant large number. The first term, which equals γ if there exists at least one task (say, task j) exceeds its deadline (i.e., $m_{i,j} > d_{i,j}$) while 0 otherwise, is the penalty of violating performance constraint. To determine this value, we need to estimate the schedule length, which depends on not only assignment but also the schedule. While we can use exhaustive search to obtain the minimum value and the corresponding task schedule, given the task allocation solution, we can resort to well-studied heuristics (e.g., [69]) to acquire an optimized task schedule. In this work, to reduce computational complexity, we use the *levels* of tasks as the task scheduling priority, that is, every time we assign the unscheduled task with the highest level to a certain processor core according to the resource binding sequence. Here, *level* of a task is computed as the total execution time along the longest path from that task to an exit task (i.e., the task with zero out-degree) [69]. As for the second term, E_i indicates the energy consumption of the task schedule in one period, being defined in Eq. (6.8). $\widetilde{R}_i$ is a function of R_i defined as follows so that low-$\widetilde{R}_i$ solutions are favored during the SA searching process. We need to introduce $\widetilde{R}_i$ rather than using R_i to represent the reliability because solutions with lower R_i are with less lifetime reliability and hence should have higher cost during the simulated annealing process.

$$\widetilde{R}_i = -\ln R_i = \sum_k (A_{i,k} \cdot \frac{L}{\max_j \{d_{i,j}\}})^{\beta_k} \tag{6.16}$$

6.4.2 Voltage Assignment for DVS-Enabled Processors

Without voltage scaling, we use Eq. (6.8) to estimate the energy consumption of a resource binding sequence (i.e., E_i), by setting all $\rho_{i,j,k}$ to 1. As applying lower voltage is usually beneficial for both energy and reliability, we apply DVS onto the processor cores as long as this feature is provided. In our algorithm, we first compute the range of possible task execution time and the associated voltage scaling parameter ρ according to the task schedule. To be specific, if a resource binding sequence can meet the performance constraint, we compute the global scale parameter based on the task schedule constructed in previous step to "stretch" all tasks uniformly, i.e.

$$\theta_i^{\mathrm{g}} = \frac{\max_j\{d_{i,j}\} - o_i}{m_i} \tag{6.17}$$

Here, o_i is the worst-case DVS overhead for the task allocation and schedule solution, depending on the number of voltage transitions and the corresponding voltage levels.

Also, we notice that some tasks can be further elongated without affecting the scheduling of any other tasks. To be specific, a task can be further extended if it finishes before the starting time of all of its successors on task graph and the starting time of the next task on the same processor core. This idle duration is referred to as slack time, denoted as $s_{i,j,k}$. We define its local scale parameter for this task as

$$\theta_{i,j,k}^{\ell} = \frac{w_{i,j,k} \cdot \theta_i^{\mathrm{g}} + s_{i,j,k}}{w_{i,j,k} \cdot \theta_i^{\mathrm{g}}} \tag{6.18}$$

Then, as in most cases a processor core can only work under a few voltage levels, we use the following procedure to determine an appropriate voltage assignment. Since the task execution time is bounded by $w_{i,j,k} \cdot \theta_i^{\mathrm{g}} \cdot \theta_{i,j,k}^{\ell}$, we choose the voltage state with "best" energy reduction and reliability enhancement (i.e., small-

est $A_{i,j,k} \times P_{i,j,k}(\rho_{i,j,k}) \times w_{i,j,k}(\rho_{i,j,k})$ for task j on processor k). After extending all tasks, we reevaluate the energy consumption of the resource binding sequence by using Eq. (6.8).

6.5 Proposed Algorithm for Multi-Mode Embedded Systems

Since the lifetime reliability constraint is a systemwide constraint (unlike the performance constraint for real-time tasks), it is not necessary to apply the same reliability constraint to every execution mode for multi-mode MPSoC embedded systems. In this section, we show how to take advantage of this flexibility to minimize energy consumption for multi-mode embedded systems. To be specific, we first generate "good" solutions in terms of reliability and/or energy for each execution mode (Section 6.5.1-6.5.3) and then we search for an optimal combination of them to obtain minimized energy while satisfying the systemwide lifetime reliability constraint (Section 6.5.4).

6.5.1 Feasible Solution Set

When we loosen the lifetime reliability constraint for a single execution mode, there are many possible task allocation and schedule solutions. However, if one solution is associated with higher energy consumption and at the same time has lower lifetime reliability when compared to another solution, it definitely should not be considered for the combination of solutions of all execution modes.

Based on the above, let us first introduce a key concept, *feasible solution set*, in our proposed methodology. Among the task allocation and schedule solutions for a certain task graph (denoted as set X), there exists a subset Y that satisfies the following two conditions.

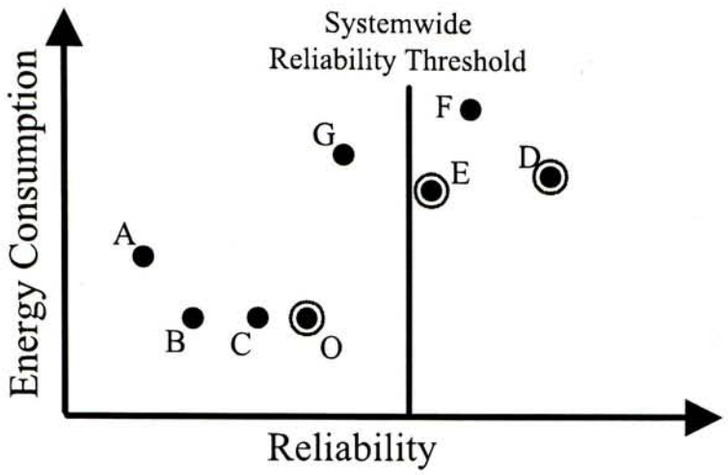


Figure 6.2: An Example of Feasible Solution Set.

Internal stability *Given two solutions $u, v \in Y$, if u consumes more energy than v, it must result in higher lifetime reliability at the target service life, and vice versa.*

External stability *For any solution $w \in X \setminus Y$, there exists at least one solution $u \in Y$ such that u consumes less energy and have higher lifetime reliability than w.*

We refer to Y as the feasible solution set, denoted as $\mathcal{F}$ in the rest of this paper. *Now, our problem of task allocation and scheduling for a single execution mode comes down to identifying $\mathcal{F}$.* Fig. 6.2 shows an example, wherein several solutions are plotted on a two-dimensional plane as points according to their lifetime reliability and energy consumption. The feasible solution set in this case is $\mathcal{F} = \{\mathbf{O}, \mathbf{D}, \mathbf{E}\}$. Consider a solution outside this set, say **G**. It implies higher energy consumption and lower lifetime reliability than solution **D**. Note that, solution **O** should be kept although it violates the global reliability constraint, because it is a possible candidate in the final combined solutions in our multi-mode systems.

6.5.2 Searching Procedure for a Single Mode

The simulated annealing-based approach presented in Section 6.4 returns a single solution with minimum energy consumption under lifetime reliability and performance constraint. To find a set of feasible solutions as candidates for later multi-mode combination, we modify the simulated annealing procedure as shown in Fig. 6.3.

In a typical SA-based algorithm, it is only necessary to keep the current solution during the searching procedure and the best solution explored so far, but in our method, every newfound solution is first checked with the cost, which is defined by Eq. (6.15), to determine whether it should be accepted (Line 7). If accepted and meeting performance constraint, depending on which feasible solution set identification strategy is chosen (detailed in Section 6.5.3), it is either added into the possible solution set $\mathcal{P}$ (Line 11) or identified together with original feasible solution set (Line 13). Note that, if the static strategy is chosen, after this searching procedure all found solutions are kept in the set $\mathcal{P}$. The static identification is conducted and yields the feasible solution set $\mathcal{F}$ (Line 16). In contrast, if we perform the dynamic identification during the searching procedure, the resulting set $\mathcal{P}$ is essentially the feasible solution set $\mathcal{F}$ (Line 18).

6.5.3 Feasible Solution Set Identification

With a number of recorded solutions obtained using the above procedure, this section is concerned with identifying the feasible solution set out of them.

Static Strategy

Let us start from a simple case in which all solutions have been found out and stored before our feasible solution set identification process (this assumption will be lifted later).

```
1   Build an initial solution X and initialize 𝒫 as {X}
2   While T > T_end
3       For the iteration i from 1 to I
4           X' ← Random_Move(X)
5           C_old ← Cal_Cost(X)
6           C_new ← Cal_Cost(X')
7           If C_new < C_old or exp((C_old − C_new)/T) > rand()
8               X ← X' // accept X'
9               If X' ∉ 𝒫 and X' meet performance constraint
10                  If static identification
11                      𝒫 ← Include(𝒫, X')
12                  Else If dynamic identification
13                      𝒫 ← Identify(𝒫, X') // see Sec. 6.5.3
14      T ← T × R_cooling

15  If static identification
16      ℱ ← Identify(𝒫) // see Sec. 6.5.3
17  Else If dynamic identification
18      ℱ ← 𝒫
```

Figure 6.3: Main Flow of Searching for Feasible Solution Set.

Before presenting the proposed approach, we first introduce some definitions. All the found task allocation and schedule solutions are examined according to the lifetime reliability model and energy model in Section 6.3, and marked on a

two-dimensional plane whose x-axis and y-axis represent the lifetime reliability at the end of target service life (provided the system remains in this mode) and the expected energy consumption, respectively. With respect to a point, we divide the plane into four domains, referred as domain *I*, *II*, *III*, and *IV*. Here, the positive x-axis and negative y-axis belong to domain *II*; while the negative x-axis and positive y-axis belong to domain *IV*. Fig. 6.4 illustrates a domain division respect to point **O**. We define *original point* as the point with the lowest energy consumption and, in case of ties, with the highest lifetime reliability on this plane, denoted as point **O**.

Theorem 5 *Point* **O** *corresponds to a feasible solution, and all other feasible solutions only exist in its domain I.*

Proof Since solution **O** consumes the least energy, there is no points falling in its domain *II* or *III*. Otherwise, the existence of such solutions implies lower energy consumption than solution **O**, which is impossible. For the solutions in its domain *IV*, since they are associated with higher energy consumption and lower reliability than **O**, they cannot be feasible. Also, solutions with the same energy consumption as that of **O** but lower lifetime reliability (e.g., point **B** and **C** in Fig. 6.4) are apparently not feasible. Therefore, all feasible solutions are in domain *I* of point **O**. □

It is also worth noting that the converse of this theorem were not true, that is, not all points in domain *I* of point **O** are feasible solutions.

We propose to sweep domain *I* with respect to point **O** in a counterclockwise manner, and check the reached points (i.e., solutions) according to the following theorem. A feasible solution set $\mathcal{F}$ is initialized as $\{\mathbf{O}\}$ before sweeping. If a reached solution is a feasible one, it is included into $\mathcal{F}$ before examining the next one.

Theorem 6 *A new solution* **N** *is a feasible one if and only if it is in domain I or III of all elements in set* $\mathcal{F}$.

Proof Suppose point **N** is feasible and it is in the domain *II* of another feasible solution $\mathbf{X} \in \mathcal{F}$, by counterclockwise sweeping we must have come across point **N** before **X** and put it into $\mathcal{F}$ already. Therefore, the supposition does not hold. Next, we assume point **N** is in the domain *IV* of any solution **X** in set $\mathcal{F}$. In this case, solution **N** costs more energy and results in lower reliability when compared to solution **X**, and hence is not a feasible solution. □

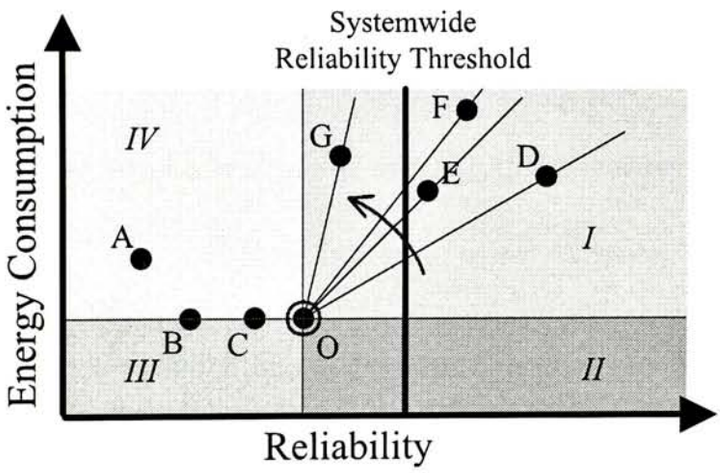


Figure 6.4: Domain Division with Respect to Solution **O**.

Consider the example shown in Fig. 6.4. By sweeping counterclockwisely, we first come across point **D**. It is in the domain *I* of point **O**, which is the only element in $\mathcal{F}$ at this moment, and therefore included into the feasible solution set. Next one is **E**. It is in domain *I* of point **O** and domain *III* of point **D**, and hence also included. Then, we come across point **F**, which is not a feasible solution since it falls into domain *IV* of point **D**. The final one is **G**. As it locates in domain *IV* of **D** and **E**, it is also not feasible. Therefore, the end result is $\mathcal{F} = \{\mathbf{O}, \mathbf{D}, \mathbf{E}\}$.

Dynamic Strategy

It is important to highlight that the above static identification strategy require to find all solutions and store them before the identification process, thus involving heavy memory overhead, especially when there are a large number of possible solutions. This problem can be avoided if we can identify feasible solutions in a random order. By doing so, feasible solution set $\mathcal{F}$ is updated whenever a newfound solution is generated, and hence we only need to maintain the current feasible solution set. In the following, a dynamic identification strategy to achieve the above objective is presented.

The feasible solution set $\mathcal{F}$ is initialized as empty. Every newfound solution is processed according to the following three rules.

Rule 1 *If the new solution is in domain I or III of ALL elements in set $\mathcal{F}$, it should be included into the feasible set $\mathcal{F}$.*

Rule 2 *If the new solution is in domain II of ANY solution* **X** *(where* $\mathbf{X} \in \mathcal{F}$*), we include the new solution into $\mathcal{F}$ and at the same time eliminate* **X** *from $\mathcal{F}$.*

Rule 3 *If the new solution is in domain IV of ANY solution* **X** *(where* $\mathbf{X} \in \mathcal{F}$*), we ignore the new solution.*

Rule 1 is consistent with Theorem 2. Rule 2 holds because the new solution must consume less energy and have higher reliability than solution **X**, if it is in domain *II* of **X**. Hence, **X** should be replaced by the new solution. As for Rule 3, given the new solution is in domain *IV* of solution **X**, it is worse than **X** in terms of both reliability and energy saving, and hence is not feasible. Note that, it is impossible that a newfound solution were marked in domain *II* of some feasible set elements and domain *IV* of some other elements simultaneously.

Consider the example shown in Fig. 6.4 again. Let us examine the dynamic identification with two random orders: **C, O, E, D, F, B, A, G** and **A, E, F, G,**

C, O, D, B. The procedures are shown in Fig. 6.5 and Fig. 6.6 respectively. Both results lead to the same feasible solution set as that obtained using static strategy (see Section 6.5.3).

original $\mathcal{F}$	new solution	updated $\mathcal{F}$
$\varnothing$	**C**	{**C**}
{**C**}	**O**	{**O**}
{**O**}	**E**	{**O, E**}
{**O, E**}	**D**	{**O, E, D**}
{**O, E, D**}	**F**	{**O, E, D**}
{**O, E, D**}	**B**	{**O, E, D**}
{**O, E, D**}	**A**	{**O, E, D**}
{**O, E, D**}	**G**	{**O, E, D**}

Figure 6.5: Identification Procedure of the First Example.

original $\mathcal{F}$	new solution	updated $\mathcal{F}$
$\varnothing$	**A**	{**A**}
{**A**}	**E**	{**A, E**}
{**A, E**}	**F**	{**A, E, F**}
{**A, E, F**}	**G**	{**A, E, F**}
{**A, E, F**}	**C**	{**C, E, F**}
{**C, E, F**}	**O**	{**O, E, F**}
{**O, E, F**}	**D**	{**O, E, D**}
{**O, E, D**}	**B**	{**O, E, D**}

Figure 6.6: Identification Procedure of the Second Example.

6.5.4 Multi-Mode Combination

Given the feasible solution set for every execution mode, we show how to combine them to achieve minimum energy consumption under systemwide lifetime reliability constraint in this subsection.

For a fixed execution probability combination, the energy consumption has been defined by Eq. (6.9). Suppose the feasible solution set $\mathcal{F}_i$ of mode i is composed of q_i elements. We denote by E_i^ℓ the energy consumption given the ℓ^{th} feasible solution ($\ell = 1, \cdots, q_i$) for execution mode i. In addition, we introduce a function $1_{\{\ell \Rightarrow i\}}$, which equals to 1 if we choose the ℓ^{th} solution from the feasible solution set $\mathcal{F}_i$ for mode i; otherwise 0. With these notations, Eq. (6.9) can be rewritten as

$$E(y_1, y_2, \cdots, y_n) = \sum_i y_i \cdot \sum_\ell E_i^\ell \cdot 1_{\{\ell \Rightarrow i\}} \tag{6.19}$$

Subject to the constraint that

$$\forall i: \qquad \sum_\ell 1_{\{\ell \Rightarrow i\}} = 1 \tag{6.20}$$

More generally, when we consider the usage information obtained from a large user group, we can use a joint probability density function $f_{Y_1, Y_2, \cdots, Y_n}(y_1, y_2, \cdots, y_n)$ to represent the probabilities that the system is in various execution modes. Thus, the optimization objective becomes to minimize the expected energy consumption over service life, that is, to minimize $\mathbf{E}[E(Y_1, Y_2, \cdots, Y_n)]$, where $\mathbf{E}[X]$ indicates the expectation of random variable X.

As the performance constraints of all execution modes have been guaranteed in the proposed searching procedure, besides Eq. (6.20) we need to consider the lifetime reliability constraint only. Similar to the energy consumption issue, with Eq. (6.14) we compute the expected value given the fixed probabilities y_i for execution mode i as

$$R(y_1, y_2, \cdots, y_n) = \exp\left[-\sum_k \left(\sum_i \frac{y_i \cdot \sum_\ell A_{i,k}^\ell \cdot 1_{\{\ell \Rightarrow i\}}}{\max_j\{d_{i,j}\}} \cdot L\right)^{\beta_k}\right] \tag{6.21}$$

where, $A_{i,k}^\ell$ represents the aging effect of task schedule ℓ on processor k in execution mode i. The reliability constraint comes down to keep the expected lifetime reliability over the execution probability distribution exceeding a threshold, i.e.,

$$\mathbf{E}[R(Y_1, Y_2, \cdots, Y_n)] \geq \eta \tag{6.22}$$

We therefore formulate the multi-mode solution combination problem as follows:

$$\begin{array}{ll} \min & \mathbf{E}[E(Y_1, Y_2, \cdots, Y_n)] \\ \text{st.} & \mathbf{E}[R(Y_1, Y_2, \cdots, Y_n)] \geq \eta \\ & \forall i : \sum_\ell 1_{\{\ell \Rightarrow i\}} = 1 \\ & \forall i, \ell : 1_{\{\ell \Rightarrow i\}} = 0 \text{ or } 1 \end{array}$$

This optimization problem can be solved quite efficiently since the number of execution modes for an embedded system is typically small.

6.6 Experimental Results

6.6.1 Experimental Setup

We conduct two sets of experiments with different task graphs and hypothetical MPSoC platforms to evaluate the proposed approach. All task graphs are generated by TGFF [32]. The power consumption of tasks on processor cores are randomly generated, while the range is set according to state-of-the-art technology [116]. Although the proposed approach is applicable for the combination of

multiple failure mechanisms, since there is no public data on the influence weight of various hard errors, we use the well-studied electromigration failure model presented in [39] for our experiments. The parameters are set to cross-sectional area of conductor $A_c = 6.4 \times 10^{-8} cm^2$, the current density $J = 1.5 \times 10^6 A/cm^2$ and the activation energy $E_a = 0.48eV$. The simulated annealing parameters are set to initial temperature 100, terminal temperature $T_{end} = 10^{-5}$, cooling rate $R_{cooling} = 0.99$, and iteration count $I = 20$.

To demonstrate the effectiveness of the proposed algorithms, we compare the proposed multi-mode approach, the proposed single-mode approach wherein all execution mode needs to satisfy the given reliability constraint, and a greedy heuristic constructed by ourselves due to the lack of prior work on the same topic. In this heuristic, we first build a task schedule to shorten the schedule length and reduce energy consumption by using list scheduling proposed in [14], an extension of classic Highest Levels First with Estimated Times (HLFET) [3] for heterogeneous systems. Note that, to take both energy consumption and performance into account, the *critical path length* of task j on processor k is redefined as $\text{cpl}'(\tau_j, p_k) = \text{cpl}(\tau_j, p_k) - P_{j,k}$ according to the heuristics in [54], where $\text{cpl}(\tau_j, p_k)$ is the critical path length of a task τ_j scheduled on processor p_k [14]. We then attempt to meet the reliability constraint in a greedy manner if the system reliability constraint is violated, that is, the processor core with the minimum lifetime reliability is selected in each iteration and the task causing the highest stress is assigned onto another core without violating performance constraints. The moved task is then locked. The procedure terminates if no more moves are available or reliability constraint can be met.

6.6.2 Case Study

We consider three task graphs as shown in Fig. 6.7 (denoted as task graph (a), (b), and (c) respectively hereafter) in the first experiment, each corresponding to a particular execution mode, and schedule them on two processor cores with failure distribution slope parameter β = 1.5, 2, respectively. The probabilities that the system is in execution modes (a)-(c) are set to be 0.3, 0.3, and 0.4, respectively. By conducting the proposed searching procedure we obtain a set of feasible solutions for each mode, sorted in the order of R_i and listed in Table 6.1. Some schedules (e.g., solution 2-7 for task graph (a)) cannot meet the lifetime reliability constraint, but they are kept because some other task graphs (e.g., task graph (b)) may provide reliability margin.

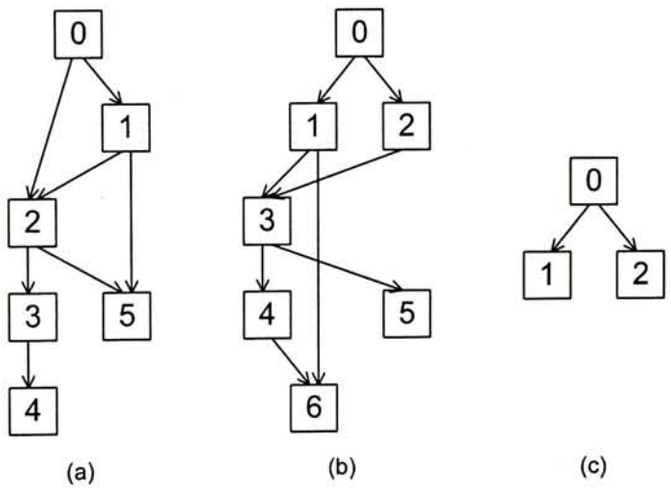


Figure 6.7: Task Graphs for an Example Multi-Mode System.

Then, by optimization, we obtain the combination with the minimum expected energy consumption 16.875 meeting the lifetime reliability requirement $R_i \geq 36.8\%$ (or e^{-1}) at the target service life $L = 10$ years [64]. We choose solution 7, 4, 5 for task graph (a), (b), and (c), respectively.

We compare this result with that obtained by the baseline greedy heuristic and the proposed single-mode algorithms. The selected solutions are illuminated in Ta-

Task Graph Label	Solution No.	R_i (%)	E_i	C_i	$\max_j\{m_{i,j}\}$	Resource Binding Sequence
(a)	1	39.16	23.179	21.550	52.835	(0, 0, 1, 0, 1, 1)
	2	**36.70**	22.992	22.851	46.742	(1, 0, 1, 0, 1, 0)
	3	**34.91**	22.370	23.344	50.028	(0, 0, 1, 1, 1, 1)
	4	**34.18**	21.312	22.686	37.021	(1, 0, 1, 0, 1, 1)
	5	**27.92**	20.503	25.939	34.214	(1, 0, 1, 1, 1, 1)
	6	**17.05**	20.061	35.188	33.807	(1, 1, 1, 0, 1, 1)
	7	**11.80**	19.253	40.793	31.001	(1, 1, 1, 1, 1, 1)
(b)	1	65.09	15.437	6.574	71.702	(1, 0, 1, 1, 1, 0, 0)
	2	59.19	15.358	7.986	78.068	(1, 1, 1, 1, 0, 0, 0)
	3	56.94	14.921	8.332	62.918	(1, 0, 1, 1, 0, 0, 0)
	4	47.54	14.910	10.993	57.684	(1, 0, 0, 1, 0, 0, 0)
	5	45.49	14.488	11.318	52.101	(1, 0, 1, 0, 0, 0, 0)
	6	**36.51**	14.477	14.466	46.867	(1, 0, 0, 0, 0, 0, 0)
(c)	1	44.05	23.036	18.726	33.275	(1, 1, 0)
	2	41.98	22.559	19.416	25.813	(0, 1, 0)
	3	39.77	19.889	18.185	27.235	(1, 0, 1)
	4	**35.33**	17.034	17.575	23.355	(1, 0, 0)
	5	**27.10**	16.556	21.437	15.892	(0, 0, 0)

R_i: lifetime reliability at the end of target service life given mode i;
E_i: energy consumption for execution mode i; C_i: cost of task schedule;

Table 6.1: Feasible Solution Set (End Result).

ble 6.2. The baseline approach cannot provide a solution meeting this requirement because of task graph (a). On the other hand, because the single-mode method does not allow the reliability constraint violation of any modes, it results in significant reliability margin. The multi-mode method, by contrast, makes full use of such margin, and therefore provides 14.1% energy saving when compared with the single-mode approach.

6.6.3 Sensitivity Analysis

It is interesting to analyze the impact of lifetime reliability threshold on the effectiveness of the proposed algorithm. As shown in Fig. 6.8, no solutions can be

Approach	Task Graph Label	$R_i(\%)$	E_i	$\max_j\{m_{i,j}\}$	Resource Binding Sequence	$\mathbf{E}[E]$
Baseline	(a)	–	–	–	–	–
	(b)	42.71	15.008	43.283	(0, 0, 0, 1, 0, 0, 0)	
	(c)	41.98	22.559	25.813	(0, 1, 0)	
Single Mode	(a)	39.16	23.179	52.835	(0, 0, 1, 0, 1, 1)	19.255
	(b)	45.49	14.488	52.101	(1, 0, 1, 0, 0, 0, 0)	
	(c)	39.77	19.889	27.235	(1, 0, 1)	
Multi Mode	(a)	11.80	19.253	31.001	(1, 1, 1, 1, 1, 1)	16.875
	(b)	47.54	14.910	57.684	(1, 0, 0, 1, 0, 0, 0)	
	(c)	27.10	16.556	15.892	(0, 0, 0)	

Table 6.2: Energy Consumption Comparison between the Single-Mode Method and the Multi-Mode Combination Approach.

achieved by using the baseline method when the reliability threshold is tight (i.e., higher than 32%), while the proposed single-mode approach results in good solutions until the reliability threshold increases to 39%. The proposed multi-mode approach is able to give solutions when the reliability threshold is as high as 49%.

If the reliability threshold can be relaxed, both the energy consumptions obtained by the single-mode method and multi-mode one decrease, and finally they converge when the threshold decreases to 11.8%. In the range 11.8–39%, the proposed approach always leads to better result than the single-mode one. The energy consumption of the baseline approach also decreases with the relaxation of reliability requirement, but it always results in the highest energy consumption among these methods. In particular, when the reliability threshold is 11.8%, the energy consumption obtained with the baseline method is 17.27. With the same energy consumption, the proposed single-mode and multi-mode approach are able to achieve much higher lifetime reliability 27% and 42% (see the black stars) after 10-year service. Generally speaking, the warrantee period for electronic products are shorter than their designed service life, and we are interested in the failure rates at this time point. Suppose our system's warrantee period is 3-year, the fail-

ure rates for these three methods are 17.9%, 15.2% and 11.7%, respectively. In other words, there are roughly 6.2% less failures with our proposed method when compared to the greedy heuristic at the end of the warrantee period.

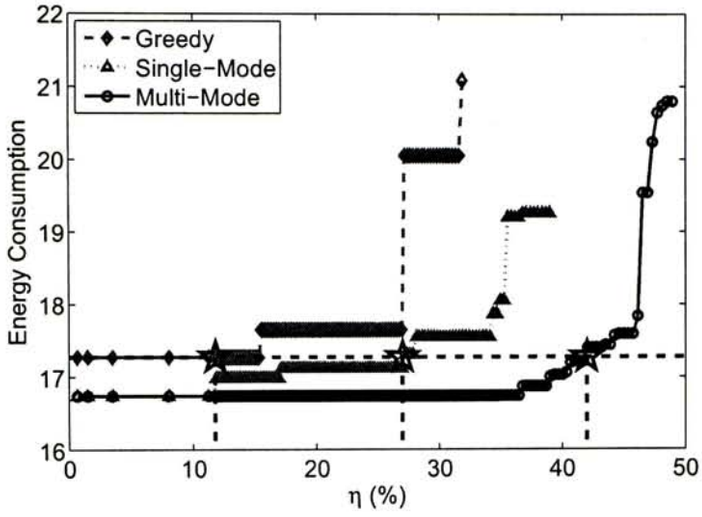


Figure 6.8: Variation in Energy Consumption with Reliability Threshold.

6.6.4 Extensive Results

In this experiment, we consider relatively large MPSoC platform and the associated task graphs, wherein we have 5 execution modes and their task quantities are 69, 7, 14, 3, and 29 (denoted as task graph (d)-(h) respectively), mapping onto a 8-core heterogeneous MPSoC. The failure distribution slope parameters β for processor cores are 2, 4, and six 1.5, respectively.

As shown in Fig. 6.9, only when the reliability threshold is lower than 29%, we can obtain solutions for all execution modes using the baseline greedy heuristic. In this range, both single-mode and multi-mode approaches provide up to 26.3% and 27.8% energy reduction when compared with the greedy solutions, respectively.

The single-mode approach is able to meet tighter reliability constraints and save more energy when compared to the greedy heuristic. But still, when the re-

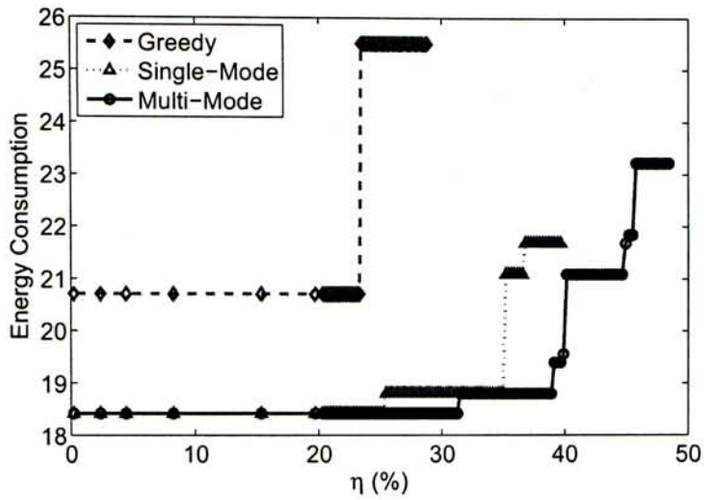


Figure 6.9: Comparison between Energy Consumption of Multi-Mode System under Constraints.

liability threshold is higher than 39.7%, there is no solution for some execution modes and hence the entire system. For instance, when threshold is 45%, no task schedule meeting both constraints can be found for modes (d) and (e). With the proposed multi-mode combination approach, however, we can obtain a solution for the entire multi-mode system. This is because some other modes, such as (g), provides sufficient reliability margin. Besides, the multi-mode approach always achieves lower energy consumption when compared with the single-mode approach, because it explores a larger solution space that includes the solution obtained by the single-mode method.

Finally, we allocate these task graphs onto a 8-core homogeneous MPSoC platform, where all processor cores have the same architecture and hence the same failure distribution (β = 1.5), and consume the same execution time and power consumption on each task. In this case, the three methods obtain solutions with the same energy consumption when the reliability constraint is set as $\eta = 27.7\%$, but the proposed multi-mode approach is able to achieve higher lifetime reliability, $\eta = 37.3\%$. This is because, for each single execution mode, all task schedules

in the feasible solution set obtained with the proposed algorithm have the same lifetime reliability and energy consumption, but different resource allocation. The proposed multi-mode combination step tends to choose the task schedules for multiple execution modes such that all processor cores have similar wear-out stress. This feature, however, cannot be exploited by the other two methods.

6.7 Conclusion

Advancement in technology has brought with ever-increasing adverse impact on the lifetime reliability of embedded MPSoCs. In this work, we propose novel task allocation and scheduling algorithms to minimize the expected energy consumption of multi-mode embedded systems under performance and lifetime reliability constraints. As shown in our experimental results, the proposed methodology is able to meet tight reliability constraints and results in significant energy savings, especially for heterogeneous multiprocessor system.

□ End of chapter.

Chapter 7

Customer-Aware Task Allocation and Scheduling

The content in this chapter has been accepted for publication in the proceedings of *ACM/IEEE Design Automation Conference (DAC)* 2011 [51].

7.1 Introduction

When designing complex embedded systems, it is increasingly popular to take a pre-designed multiprocessor system-on-a-chip (MPSoC) platform (e.g., ARM PrimeXsys platform [7]) and map applications onto it. A critical step in the platform-based embedded system design flow is the so-called task allocation and scheduling (TAS) process, which determines the task mappings to processing elements (PEs) and their execution sequences to meet design specifications. Among the various optimization objectives during the TAS process, energy consumption is one of the primary concerns, especially for handhold embedded systems. Based on the widely-used dynamic voltage frequency scaling (DVFS) techniques [58], there is a rich literature on energy-efficient task allocation and scheduling techniques

(e.g., [61]). Meanwhile, due to the everincreasing temperature and power density of integrated circuits (ICs), the lifetime reliability of MPSoC products caused by wearout effects has also become a serious concern [16, 64, 95]. To tackle this problem, some recent work (e.g., [52]) proposed to explicitly take lifetime reliability into consideration during the TAS process.

Electronic products (e.g., smart phones) are seldom designed exclusively for one or a few specific customers. In most cases, a great volume of products are manufactured according to an identical design and shipped to different customers. The products, since bought by end users, however, start to experience different life stories. For instance, some users might mainly use their smart phones for making calls, while some others use it for music play most of the time. In other words, customers may have significant different usage strategies for the same system. We refer to this phenomenon as the *usage strategy deviation* of the product.

The allocation and scheduling of tasks on embedded systems at design stage, albeit carefully conducted, at best can be optimized for a hypothetical common case. It is very likely that the obtained TAS solution is not energy-efficient or reliable from particular customers point of view, due to the fact that their usage strategy deviates significantly from the "common case". By lifting the implicit assumption of applying the unified task schedules on all products, a *personalized* TAS solution for each individual product can be more energy-efficient and/or reliable.

Motivated by the above, we propose to design the product in such manner that it has the capability of extracting its own usage strategy and adjusting the task schedules at runtime when necessary. The customer-aware task schedules for multi-mode MPSoC embedded systems are constructed as follows. First, we generate an initial TAS solution for each execution mode such that the expected energy consumption is minimized under a given lifetime reliability constraint. Sequen-

tially, we conduct online adjustment for each particular survival chip at regular interval based on its past usage strategy to guarantee its lifetime reliability and/or reduce its energy consumption. The performance overhead for online adjustment is negligible because the aging effect of IC products is typically in range of years and hence the adjustment interval can be set in the range of days or months. To have more flexibility during online adjustment, however, certain tasks may need to be mapped onto different types of PEs, which requires more design effort and storage space. This is taken into consideration in our solution as a constraint. Experimental results on hypothetical MPSoC products with various task graphs show that the proposed solution is able to significantly increase their lifetime reliability and is also helpful for energy reduction.

The remainder of this paper is organized as follows. Section 7.2 reviews related previous work and formulates the problem investigated in this paper. Section 7.3 presents the proposed simulated annealing (SA) based TAS technique used at design stage. Then, the reliability model for online adjustment and the corresponding online adjustment technique are introduced in Section 7.4. Experimental results on hypothetical multi-mode MPSoCs are presented in Section 7.5. Finally, Section 7.6 concludes this work.

7.2 Prior Work and Problem Formulation

7.2.1 Related Work and Motivation

Due to the ever-increasing adverse impact of relentless technology scaling on IC lifetime reliability, it is essential to take circuit aging effect into consideration during the task allocation and scheduling process. [52] first addressed this problem by generating a unique task schedule with maximum lifetime reliability for a single-mode embedded system. Later, the same authors extended their work in [49] for

multi-mode MPSoCs, considering to minimize energy consumption under a given lifetime reliability constraint. The problem is solved in two phases: a set of possible task schedules are generated for each execution mode, and then a multi-mode combination algorithm is used to select a schedule from each set for global optimization.

In the above works, TAS solutions are generated at design stage and a unified task schedule for each execution mode is constructed for all the products. As discussed earlier, in the presence of customers' usage strategy deviation, the task schedules can be optimized at best with respect to a usage strategy distribution $f_{\mathbf{Y}}(\mathbf{y})$, obtained via market research or customer survey. In other words, setting energy savings as the optimization objective, we can construct task schedules to minimize the *expected* energy consumption of all products and guarantee that the *expected* lifetime reliability of all products is no less than a predefined threshold. The resulting unified schedule, however, may not perform well from particular customers' point of view.

To take an example, let us consider a simple MPSoC product with three execution modes and two processor cores. The task graphs are generated by TGFF [32] and demonstrated in Fig. 7.1. The usage strategy distribution is set to $y_1 \sim N(0.5, 0.5^2)$, $y_2 \sim N(0.5, 0.5^2)$, and $y_3 = 1 - y_1 - y_2$, provided all variables are in the range between 0 and 1. We apply the task schedules that minimize the expected energy consumption over usage strategy distribution under constraints onto a volume of products with the same design, obtained by using the technique proposed in [49]. The measurements of 10,000 sample products are plotted in Fig. 7.2. As shown in this figure, because of the usage strategy deviation, although the reliability threshold is set to 47%, the reliability of sample products ranges from 35% to 65%. In particular, around half of products (the marks below red line) are not reliable as expected, while the remaining have too much reliability margin that can be used

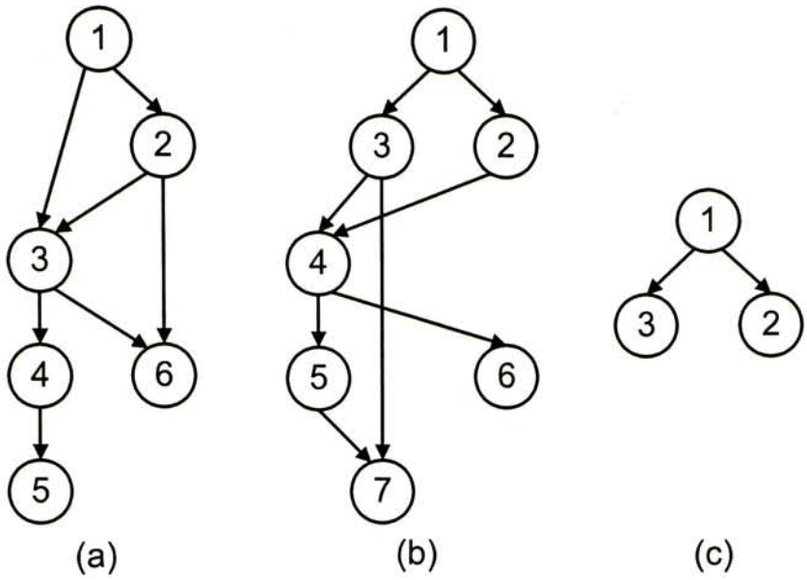


Figure 7.1: Task Graphs in the Example.

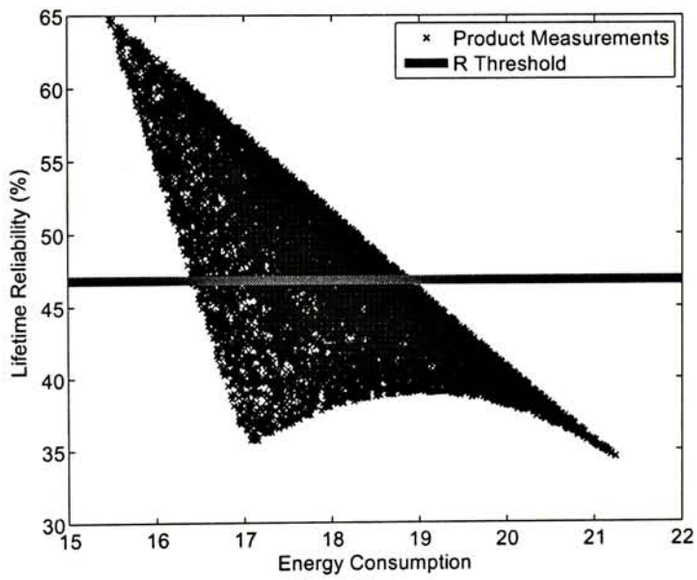


Figure 7.2: Impact of Usage Strategy Deviation.

to reduce energy.

The above observation motivates the proposed customer-aware task allocation and scheduling technique in this paper.

7.2.2 Problem Formulation

To achieve the objective of this work, we need to generate an initial TAS solution at design-stage and then conduct online adjustment after the product is deployed to the market. Consequently, we partition the problem into two phases and formulate them separately in this section.

Problem 1 [Design Stage]: Given

- q execution modes. A directed acyclic task graph $\mathcal{G}^k = (\mathcal{T}^k, \mathcal{E}^k)$ for mode k, wherein each node in $\mathcal{T}^k = \{\tau_i^k : 1 \leq i \leq n^k\}$ represents a task in $\mathcal{G}^k$, and $\mathcal{E}^k$ is the set of directed arcs which represent precedence constraints. Each task graph $\mathcal{G}^k$ has a deadline d^k;
- The joint probability density function[1] that the system is in various modes $f_{\mathbf{Y}}(\mathbf{y})$, where y_k represents the probability that the system is in execution mode k;
- A platform-based MPSoC embedded system that consists of a set of processors $\mathcal{P} = \{P_j : 1 \leq j \leq m\}$, belonging to $\mathcal{V}$ categories;
- Execution time table $\{c_{k,i,j} : 1 \leq k \leq q, 1 \leq i \leq n^k, 1 \leq j \leq m\}$, where $c_{k,i,j}$ is the execution time of task τ_i^k on processor P_j. If a task cannot be executed by a certain processor, the corresponding $c_{k,i,j}$ is set to infinity;
- Power consumption table $\{p_{k,i,j} : 1 \leq k \leq q, 1 \leq i \leq n^k, 1 \leq j \leq m\}$, where $p_{k,i,j}$ is the power consumption of τ_i^k on processor P_j;
- The target service life t_L and the corresponding reliability requirement $\eta\%$;

To determine a periodical task schedule for each execution mode such that the expected energy consumption over all products is minimized under the per-

[1]The mode execution probabilities can be estimated as in [49, 85].

formance constraints that all tasks are finished before deadlines and the reliability constraint that the expected reliability at the target service life is no less than $\eta\%$.

With the initial task schedules generated at the design stage, for a particular MPSoC product, its task schedule can be online adjusted at regular intervals. The problem to be solved at the end of the u^{th} interval is formulated as

Problem 2 [Online Adjustment]: Given all parameters as specified in Problem 1, and

- Interval length t_I;
- Usage strategy $\mathbf{y}_\ell$ of the ℓ^{th} interval for $1 \leq \ell \leq u$;
- Task mapping flexibility constraints;

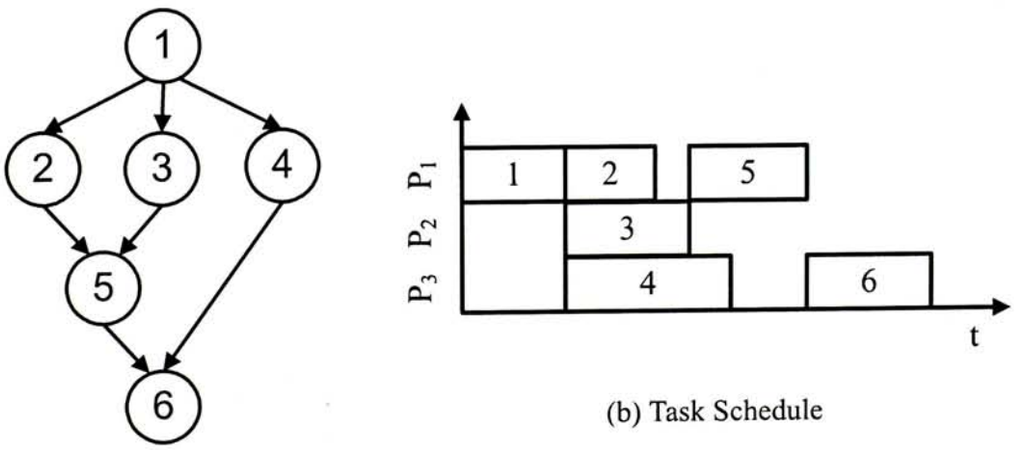
To determine a periodical task schedule for each execution mode such that the energy consumption of this particular product is minimized under the performance and reliability requirement. It is worth to note that the lifetime reliability of the MPSoC products is a statistical value and the reliability constraint here is set as same as that in Problem 1, i.e., the percentage of products that survive until the target service life is no less than $\eta\%$.

7.3 Proposed Design-Stage Task Allocation and Scheduling

We propose to handle Problem 1 with a simulated annealing-based algorithm. Since it is impossible to generate task schedules for each individual chip, the optimization objective is the *expected* energy consumption over all the products.

7.3.1 Solution Representation and Moves

An effective solution representation and the corresponding moves is very important for any SA-based technique. To the best of our knowledge, all existing techniques (e.g., [71]) do not have a 1-to-1 correspondence between the representation and the task schedule. Consider the one proposed in [52], wherein a schedule is represented as (*scheduling order sequence*; *resource assignment sequence*) together with three kinds of moves. While it has been proved that all possible task schedules are reachable starting from an arbitrary initial valid one [52], this representation suffers from redundancy problem. To clarify, we consider the task graph shown in Fig. 7.3(a). Based on the solution representation (1, 2, 3, 4, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3), which means task 1 is scheduled on P_1 first, following by task 2, 3, 4, 5, and 6, on P_1, P_2, P_3, P_1, and P_3 respectively, we can reconstruct the unique task schedule, demonstrated in Fig. 7.3(b). However, some solutions, such as (1, 3, 2, 4, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3) and (1, 4, 3, 2, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3), correspond to the same schedule. In other words, some moves cannot lead to new schedules, which can result in significantly adverse impact on the efficiency of searching process.



(a) Task Graph

(b) Task Schedule

Figure 7.3: An Example of Solution Representation.

This observation motivates us to propose a novel solution representation, which has 1-1 correspondence property between task schedules and solution representations. Given a solution representation, if swapping any two tasks in a subsequence of the scheduling order sequence does not change the schedule, the tasks in the subsequence are referred to as *interchangeable*. In other words, swapping the order of these tasks in the scheduling order sequence cannot result in a new schedule. In our example, given the solution (1, 2, 3, 4, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3), tasks 2, 3, and 4 form such a subsequence. In addition, tasks 4 and 5 form another one. The task allocation and scheduling for the MPSoC is therefore represented as (*scheduling order sequence*; *resource assignment sequence*), and the interchangeable tasks are marked with square bracket. In addition, in case that two adjacent tasks in the scheduling order sequence are interchangeable, we always keep the one with smaller index in the front. For example, the schedule shown in Fig. 7.3(b) is represented as (1, [2, 3, [4], 5], 6; P_1, P_1, P_2, P_3, P_1, P_3). Clearly, a solution representation is regarded as a *valid* one if (i). the scheduling order sequence conforms to the partial order designated by the task graph and (ii). if two adjacent tasks are interchangeable, the one with smaller index is kept in the front.

Before the definition of the moves, we need to introduce a few concepts. With the above representation, we define *zones* in the scheduling order sequence. The tasks in the scheduling order sequence are swept from left to right. A new zone appears when we reach a task that does not belong to any subsequence or we reach a new subsequence, and ends when we move to the next task in the first case or the subsequence terminates in the second case. An example is demonstrated in Fig. 7.4, where four zones are identified. The tasks are labeled with the zone indexes. Note that, in case that a task belongs to two zones, we always label it with the smaller index. In our example, task 4 belongs to two zones and is labeled as a task in zone 2. To change the scheduling order sequence, we can randomly

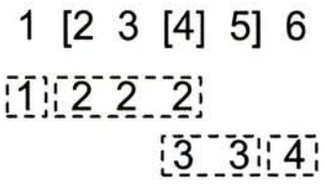


Figure 7.4: Zone Representation.

pick up a task and refer to it as *source task*. Given the source task is in zone i, if there exists a task just precedent to zone i in the scheduling order sequence, it is referred to as the *sink task* with respect to the source. To take examples, suppose task 3 is selected as the source, it belongs to zone 2 and hence its sink is task 1. Given the source task 4, although it belongs to two zones, it is referred to as a task in zone 2 and therefore its sink is task 1. Similarly, the source task 5 corresponds to the sink task 3.

Now we are ready to introduce the moves that guarantee the completeness of searching process. Two kinds of moves are defined as follows:

- M1: Pick up a task in the scheduling order sequence as the source and insert it in the front of its sink, if there is no precedence constraint between the source and sink tasks.
- M2: Change the resource assignment of a task.

To move to a neighbor solution, we start with the original solution and randomly apply a move defined above. Sequentially, we need to conduct a post-processing step to identify the interchangeable subsequences. This is easily achievable because two adjacent tasks in the scheduling order sequence are interchangeable if and only if (i). there is no precedence constraint between them and (ii). they are assigned onto different processor cores. In addition, we move the tasks with small indexes forward as much as possible. The proof of completeness and 1-1 correspondence are presented in Appendix.

To reconstruct the schedule with a given solution representation, we assign the tasks according to the scheduling order sequence iteratively. Every time the i^{th} task in the scheduling order sequence whose index is j is picked up and its core assignment is identified by the j^{th} element in the resource binding sequence. We simply set its starting time as the earliest available time.

7.3.2 Cost Function

Since the objective is to minimize the expected energy consumption under the performance and reliability constraints, the cost function used in our SA-based technique consists of three terms, each for an objective or a constraint.

$$\text{Cost} = \mathbf{E_Y}[\text{Energy}] + \mu \cdot \mathbf{I}[\exists i,k : e_i^k > d^k] + \mu \cdot \mathbf{I}[R^{sys}(t_L) < \eta\%] \tag{7.1}$$

where $\mathbf{E_Y}$ indicates the expectation over usage strategy distribution $\mathbf{Y}$, μ is a significantly large number, and the indicator function $\mathbf{I}[A]$ equals to 1 if event A is true while 0 otherwise. Thus, the large cost μ is the penalty of violating constraints.

In case that no DVFS technique is enabled, the computation of energy consumption in an execution mode k is a trivial problem. We can simply sum up the energy consumption of all tasks, namely,

$$\text{Energy}_k = \sum_{i=1}^{n^k} \sum_{j=1}^{m} \mathbf{I}[i \rightarrow j] \cdot c_{k,i,j} \cdot p_{k,i,j} \tag{7.2}$$

Thus, given the usage strategy $\mathbf{y}$, the energy consumption of a system can be expressed as

$$\text{Energy} = \sum_{k=1}^{q} \sum_{i=1}^{n^k} \sum_{j=1}^{m} \mathbf{I}[i \rightarrow j] \cdot c_{k,i,j} \cdot p_{k,i,j} \cdot y_k \tag{7.3}$$

Nevertheless, it is difficult to compute the expected energy consumption $\mathbf{E_Y}[\text{Energy}]$ according to its closed-form expression, i.e.,

$$\mathbf{E}_{\mathbf{Y}}[\text{Energy}] = \int \cdots \int_{\mathbf{Y}} \text{Energy} \cdot f_{\mathbf{Y}}(\mathbf{y}) d\mathbf{y} \tag{7.4}$$

In this work, we propose to discretize the range of y_k (i.e., [0,1]) into partitions of u equal intervals. As a consequence, since the state space of $\mathbf{y}$ is q-dimensional, there are u^q partitions in total. We compute the energy consumption with respect to a sample usage strategy with Eq. (7.3) for each partition and sequentially approximate the expected value with the weighted sum.

Given the task schedule, the second term in Eq. (7.1) can be determined by comparing the finish time e_i^k and deadline d_i^k of tasks and it is independent of usage strategy.

We then move to discuss the third term. We assume the reliability function of a processor core follows Weibull distribution [52], where the shape parameter β reflects the architecture of processor core while the scale parameter θ indicates the rate of suffering from aging effect and hence depends on operational temperature, supply voltage, and frequency.

$$R(t) = \exp\left(-\left(\frac{t}{\theta}\right)^{\beta}\right) \tag{7.5}$$

Let $\Omega(s^k)$ be the aging rate of the processor in execution mode k [52], induced by the task schedule s^k. This quantity indicates the rate of a processor core suffering from aging effect. With the similar argument in [46], we can express the expected aging rate as

$$\Omega(\mathbf{s}) = \sum_{k=1}^{q} \Omega(s^k) \cdot y^k \tag{7.6}$$

and approximate the reliability function as

$$R^{sys}(t_L) = \exp\left(-\sum_{j=1}^{m} \left(\frac{t_L}{\Omega(\mathbf{s})}\right)^{\beta_j}\right) \tag{7.7}$$

Similar to the expected energy consumption, $R^{sys}(t_L)$ is a function of usage strategy. We therefore use the approximated value obtained by discretization for cost evaluation.

7.3.3 Impact of DVFS

For energy saving, DVFS techniques are enabled on most MPSoC designs to make use of the task slack time for energy consumption reduction, and lifetime reliability enhancement. The proposed cost function should be able to capture these impacts in case that DVFS is applied. We therefore present the modification on the cost function in this section.

We define the voltage scaling parameter ρ as a value within the range $[0, 1]$, where $\rho = 1$ implies the processor is working under maximum supply voltage. The scaled supply voltage is therefore ρV_{dd}.

Without DVFS, the power dissipation of a processor core can be described by the following equation [59]

$$\begin{aligned} p &= P_{\text{on}} + P_{\text{dynamic}} + P_{\text{leakage}} \\ &= P_{\text{on}} + C_{\text{eff}} \cdot V_{\text{dd}}^2 \cdot f + (V_{\text{dd}} \cdot I_{\text{subn}} + |V_{\text{bs}}| \cdot I_{\text{j}}) \end{aligned} \tag{7.8}$$

where, P_{on} is the inherent power cost for keeping the processor on, which can be assumed as a constant value [59]; P_{dynamic} represents the dynamic power consumption, which is quadratic dependent on supply voltage V_{dd} and proportional to operating frequency f and the effective switching capacitance C_{eff}; P_{leakage} indicates the leakage power dissipation due to subthreshold leakage current I_{subn} and reverse bias junction current I_{j}. V_{bs} is the body bias voltage. I_{subn} is a voltage-dependent parameter, given by

$$I_{\text{subn}} = K_1 e^{K_2 V_{\text{dd}}} \tag{7.9}$$

where, K_1 and K_2 are constant fitting parameters. I_j can be approximated as a constant value, as pointed out in [74].

The operational frequency of a processor core is bounded by its slowest path, whose delay can be expressed in terms of supply voltage V_{dd}, threshold voltage V_{th}, and logic depth of the critical path L_d [74], that is,

$$\tau = \frac{L_d K_3}{(V_{\text{dd}} - V_{\text{th}})^{\kappa}} \tag{7.10}$$

where, κ is a material-related constant. K_3 is a constant related to process technology. As the operational frequency is inversely proportional to circuit delay, we have

$$f = \frac{1}{\tau} \tag{7.11}$$

Given the voltage scaling parameter ρ, substituting the scaled supply voltage ρV_{dd} into Eq. (7.10) and Eq. (7.11) yields the scaling operating frequency, i.e.,

$$f(\rho) = \frac{(\rho V_{\text{dd}} - V_{\text{th}})^{\kappa}}{L_d K_3} \tag{7.12}$$

The power consumption of a task (that is, Eq. (7.8)) with voltage scaling parameter ρ is then redefined as

$$p(\rho) = C_{\text{eff}} \cdot \rho^2 V_{\text{dd}}^2 \cdot f(\rho) + \rho V_{\text{dd}} \cdot K_1 e^{K_2 \cdot \rho V_{\text{dd}}} + |V_{\text{bs}}| \cdot I_j + P_{\text{on}} \tag{7.13}$$

It can be also normalized by calculating the effective switching capacitance C_{eff} with the condition that $p(\rho)|_{\rho=1} \equiv p$. The material and technology-related parameters in this model (e.g., K_1) are set according to [74] with $70nm$ technology.

Since DVFS provides benefits to both energy saving and reliability enhancement, for a given task schedule we reduce its ρ as much as possible. To be specific, we extend the execution time of a task to $c \cdot d^k$/makespan, where makespan is the time interval that all periodical tasks need to finish their execution once. By the normalization that $c(\rho)|_{\rho=1} \equiv c$, the execution time with DVFS can be expressed as

$$c(\rho) = \frac{(V_{dd} - V_{th})^\kappa}{(\rho V_{dd} - V_{th})^\kappa} \cdot c \tag{7.14}$$

We combine this equation with the following one to compute ρ.

$$c(\rho) = \frac{c \cdot d^k}{\text{makespan}} \tag{7.15}$$

These parameters $p(\rho)$ and $c(\rho)$ are substituted into the cost function presented in Section 7.3.2.

7.4 Proposed Algorithm for Online Adjustment

Given an MPSoC product, its usage strategy becomes available at run time, which enables us to adjust the task schedules correspondingly. Since the aging effect of IC product, as a general rule, is a slow process of years, there is no need of rapid-response capability. We therefore propose to perform the task schedule adjustment at regular intervals and each interval can be in range of days or months. In this sense, the online adjustment can be regarded as a special task of the embedded system and executed regularly. The detailed adjustment strategy is presented in this section.

7.4.1 Reliability Requirement for Online Adjustment

While a lifetime reliability model for MPSoC embedded system has been proposed in [52], but it is not readily applicable for online lifetime reliability evaluation because it cannot describe the design-stage reliability requirement given a certain product survives until time t without any information on other products. With the information of task schedules and usage strategy in the past u intervals, we are interested in the (conditional) reliability at the target service life t_L for the online decision making. Naturally, $t_L > u \cdot t_I$, because no adjustment is required after the target service life.

Given the product has been utilized for u intervals, the estimation of the reliability at time t_L depends on not only the past history but also the prediction for the future. Since the usage strategy for the entire service life is not available during usage, we need to infer the future usage strategy (denoted by $\mathbf{y}$) from the traced values. We suggest a forgetful scheme to highlight the recent usage preference: $\mathbf{y}$ is initialized as $\mathbf{y}_1$ at the end of the first interval. Since then, at the end of interval ℓ it is updated to $(1-\alpha)\cdot\mathbf{y}_\ell + \alpha\cdot\mathbf{y}$, where α is a small number. In other words, $\mathbf{y}$ has the form

$$\mathbf{y} = (1-\alpha)\cdot\mathbf{y}_u + (1-\alpha)\cdot\alpha\cdot\mathbf{y}_{u-1} + \cdots + \alpha^{u-1}\mathbf{y}_1 \tag{7.16}$$

In addition, we assume the new generated schedules $\mathbf{s}$ is always used in the rest of service life. Denoting by $\mathbf{s}_\ell$ the task schedules in the ℓ^{th} interval, we represent the reliability requirement for online adjustment as

$$R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) \geq \eta\% \tag{7.17}$$

Although we conduct the u^{th} online adjustment for the products that survive at time $u \cdot t_I$ only, the above requirement is consistent with the reliability constraint specified in Problem 1. To clarify, we denote by $R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)$ the

probability of a product surviving at time $u \cdot t_I$. Given N products with identical usage strategy and task schedules where N is a sufficient large number, the quantity of faulty products at time $u \cdot t_I$ is given by $N \cdot (1 - R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u))$, as shown in Fig. 7.5. Thus, to meet the reliability requirement $R^{sys}(t_L) \geq \eta\%$, the conditional reliability of a product at the target service life given its survival at time $u \cdot t_I$ should be no less than the ratio between good chips at the target service life and that at the end of u intervals, i.e.,

$$R_c(t_L | u \cdot t_I) \geq \frac{N \cdot \eta\%}{N \cdot R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)} \tag{7.18}$$

By definition, the conditional reliability is given by

$$R_c(t_L | u \cdot t_I) = \frac{R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)}{R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)} \tag{7.19}$$

Substituting Eq. (7.19) into Eq. (7.18) results in the reliability requirement for online adjustment, namely, Eq. (7.17).

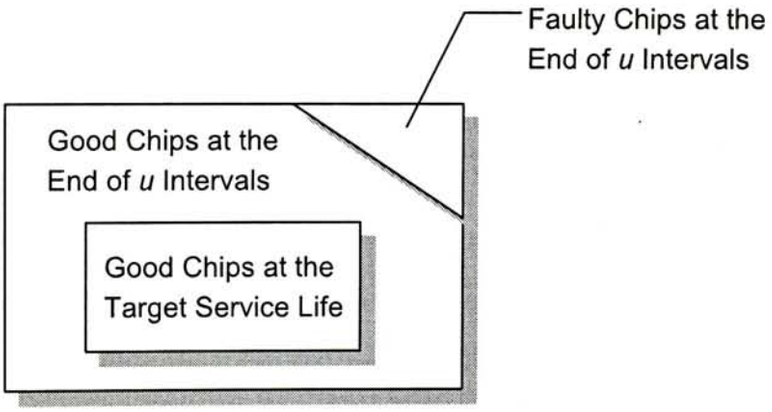


Figure 7.5: Conditional Reliability.

7.4.2 Analytical Model

With the above definition, we then move to the analytical modeling of the quantity $R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)$ in the following.

We use subscript to indicate the intervals and rewrite Eq. (7.6) as

$$\Omega(\mathbf{s}_1) = \sum_{k=1}^{q} \Omega(s_1^k) \cdot y_1^k \tag{7.20}$$

With this equation, the reliability at the end of the first interval is given by

$$R(t_I | \mathbf{y}_1, \mathbf{s}_1) = \exp\big(-(\frac{t_I}{\Omega(\mathbf{s}_1)})^{\beta}\big) \tag{7.21}$$

By the continuity of reliability function, we obtain the reliability at time $u \cdot t_I$, namely,

$$R(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) = \exp\big(-(\sum_{\ell=1}^{u} \frac{t_I}{\Omega(\mathbf{s}_\ell)})^{\beta}\big) \tag{7.22}$$

By the similar argument, we can estimate the reliability at the target service life by

$$R(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) = \exp\big(-(\frac{t_L - u \cdot t_I}{\Omega(\mathbf{s})} + \sum_{\ell=1}^{u} \frac{t_I}{\Omega(\mathbf{s}_\ell)})^{\beta}\big) \tag{7.23}$$

where, $\Omega(\mathbf{s})$ is the future aging rate, induced by task schedules $\mathbf{s}$.

Without redundant cores, the entire system functions if all processor cores are functioning. To differentiate the embedded cores, we use the subscript to represent the core index. For example, $R_j(t)$ is the reliability function of core j, and $\Omega_j(\mathbf{s}_\ell)$ is the expected aging rate of processor P_j in the ℓ^{th} interval. The system reliability is therefore given by

$$R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) = \exp\big(-\sum_{j=1}^{m}(\frac{t_L - u \cdot t_I}{\Omega_j(\mathbf{s})} + \sum_{\ell=1}^{u} \frac{t_I}{\Omega_j(\mathbf{s}_\ell)})^{\beta_j}\big) \tag{7.24}$$

label	task #	edge #	in-degree mean/std	out-degree mean/std	task # on longest path
$\mathcal{G}_a$	6	7	1.17 / 0.75	1.17 / 0.98	4
$\mathcal{G}_b$	7	8	1.14 / 0.69	1.14 / 0.90	4
$\mathcal{G}_c$	3	2	0.67 / 0.58	0.67 / 1.15	2
$\mathcal{G}_d$	31	40	1.29 / 0.81	1.29 / 1.17	10
$\mathcal{G}_e$	69	100	1.45 / 0.99	1.45 / 1.17	13
$\mathcal{G}_f$	152	233	1.53 / 0.79	1.53 / 0.74	28

Table 7.1: Description of Task Graphs.

label	processor cores		
	type	quantity	β
$\mathcal{P}_1$	v_1, v_2	1,2	2,1.5
$\mathcal{P}_2$	v_1, v_2, v_3	1,1,4	2.5,2,1.5

Table 7.2: Description of MPSoCs.

7.4.3 Overall Flow

We resort to the similar technique proposed in Section 7.3.1, while the differences stay particularly in the cost function. Since the online adjustment targets a specific MPSoC product, we need to calculate the energy consumption and reliability with respect to its usage strategy rather than the expected values over all products. In addition, as mentioned above, the lifetime reliability requirement depends on the usage strategies and task schedules in the past history and prediction for the future. Accordingly, Eq. (7.1) is rewritten as

$$\begin{aligned} \text{Cost} = {} & \mathbf{E}_{\mathbf{y}}[\text{Energy}] + \mu \cdot \mathbf{I}[\exists i,k : e_i^k > d^k] \\ & + \mu \cdot \mathbf{I}[R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) \geq \eta\%] \end{aligned} \tag{7.25}$$

label	initial solution			
	$\mathbf{E}_{\mathbf{Y}}$[Energy]	C_N^{init} (%)	C_M^{init} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}$	**16.8221**	**54.5**	**45.5**	**16.3751**
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}$	17.4577	61.6	38.4	16.8657
$\{\mathcal{G}_a, \mathcal{G}_b\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}^{\text{s}}$	22.1729	60.3	39.7	22.4243
$\{\mathcal{G}_a, \mathcal{G}_b\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}^{\text{s}}$	22.2991	32.5	67.5	22.5677
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_d\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}$	19.5647	45.9	54.1	20.0546
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_d\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}$	18.4024	47.4	52.6	18.4549
$\{\mathcal{G}_a, \mathcal{G}_e, \mathcal{G}_f\}, \mathcal{P}_2, \mathcal{U}_{\text{I}}$	20.9612	20.9	79.1	21.2033
$\{\mathcal{G}_a, \mathcal{G}_e, \mathcal{G}_f\}, \mathcal{P}_2, \mathcal{U}_{\text{II}}$	20.7217	15.3	84.7	20.8443

C_N^{init}: sample products that cannot meet reliability constraint with initial solution;
C_M^{init}: sample products that meet reliability constraint with initial solution;
$\mathbf{E}_{C_M^{\text{init}}}$[Energy]: expected energy consumption of sample products that meet reliability constraint with initial solution;

Table 7.3: Effectiveness of The Proposed Strategy.

label	online adjustment				ΔC_M (%)
	C_N^{on} (%)	C_M^{on} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]	$\mathbf{E}_{C_M^{\text{on}}}$[Energy]	
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}$	**1.2**	**98.8**	**16.0905**	**17.6617**	**53.3**
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}$	4.1	95.9	16.4147	18.5707	57.5
$\{\mathcal{G}_a, \mathcal{G}_b\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}^{\text{s}}$	28.3	71.7	22.2263	22.5648	32.0
$\{\mathcal{G}_a, \mathcal{G}_b\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}^{\text{s}}$	21.1	78.8	22.3834	22.5153	11.3
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_d\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}$	14.4	85.6	19.3790	19.8103	31.5
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_d\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}$	1.3	98.7	18.4124	18.6625	46.1
$\{\mathcal{G}_a, \mathcal{G}_e, \mathcal{G}_f\}, \mathcal{P}_2, \mathcal{U}_{\text{I}}$	0	100	21.0301	20.9117	20.9
$\{\mathcal{G}_a, \mathcal{G}_e, \mathcal{G}_f\}, \mathcal{P}_2, \mathcal{U}_{\text{II}}$	0	100	20.5244	20.5135	15.3

C_N^{on}: sample products that cannot meet reliability constraint with online adjustment;
C_M^{on}: sample products that meet reliability constraint with online adjustment;
$\mathbf{E}_{C_M^{\text{on}}}$[Energy]: expected energy consumption of sample products that meet reliability constraint with online adjustment;

Table 7.4: Effectiveness of The Proposed Strategy (Cont.).

7.5 Experimental Results

7.5.1 Experimental Setup

To demonstrate the effectiveness of the proposed technique, we setup our experiments as follows. The task graphs are generated by TGFF [32], whose attributes

are described in Table 7.1. These task graphs are scheduled on the hypothetic MP-SoCs, whose description is shown in Table 7.2. The power consumption values are randomly generated while the range is set according to state-of-the-art processors (e.g., IBM PowerPC 750CL [56]). Although the proposed analytical model is applicable for any failure mechanisms or their combinations, because of the lack of public data on the relative weights of different failure mechanisms, we consider a widely-used electromigration model [39] in our experiments. Further, the simulated annealing parameters are set to: initial temperature 10^2, end temperature 10^{-3}, iteration number 10^3, and cooling rate 0.8.

In addition, we assume most variables y_i in the joint probability density functions that used to describe the usage strategies follow truncated Gaussian distribution, as shown in Fig. 7.6. In Fig. 7.6(a), $y_1 \sim N(0.5, 0.5^2)$ and $y_2 \sim N(0.5, 0.5^2)$. The conditional domain is set to $\mathbb{D} = \{0 \leq y_1 \leq 1, 0 \leq y_2 \leq 1, 0 \leq y_1 + y_2 \leq 1\}$ and $y_3 = 1 - y_1 - y_2$. In Fig. 7.6(b), the probabilities of the product being in execution modes follow $y_2 \sim N(0.25, 0.5^2)$ and $y_3 \sim N(0.75, 0.5^2)$, and $y_1 = 1 - y_2 - y_3$. In addition, we assume two usage strategies for the MPSoCs with two execution modes: (i). $\mathcal{U}_{\mathrm{I}}^{\mathrm{s}}$: $y_1 \sim N(0.5, 0.5^2)$ and $y_2 = 1 - y_1$ and (ii). $\mathcal{U}_{\mathrm{II}}^{\mathrm{s}}$: $y_2 \sim N(0.25, 0.5^2)$ and $y_1 = 1 - y_2$, provided $0 \leq y_1 \leq 1$ and $0 \leq y_2 \leq 1$.

For each case of our experiments, 1,000 sample products are generated, whose usage strategies are independent identically distributed samples of $f_{\mathbf{Y}}(\mathbf{y})$. We use an acceptance/rejection method for generating truncated random variables $\mathbf{y}_i$ in the usage strategy distribution $f_{\mathbf{Y}}(\mathbf{y})$. To be specific, we first generate two independent random numbers a_1 and a_2, following uniform distribution $U(0,1)$. With these two numbers,

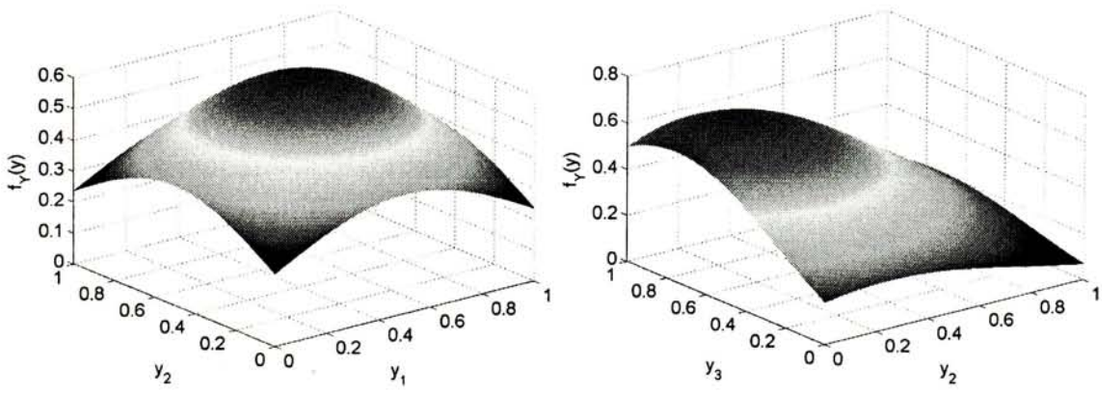
$$x_1 = \sqrt{-2\log(a_1)}\cos(2\pi a_2)$$

$$x_2 = \sqrt{-2\log(a_1)}\sin(2\pi a_2)$$

are independent and follow $N(0,1)$ [37]. Thus, since $X_i \sim N(0,1)$, the variable Y_i, which is defined as $\sigma_i \cdot X_i + \mu_i$, follows $N(\mu_i, \sigma_i^2)$. The interval length t_I is set to $10\%t_L$.

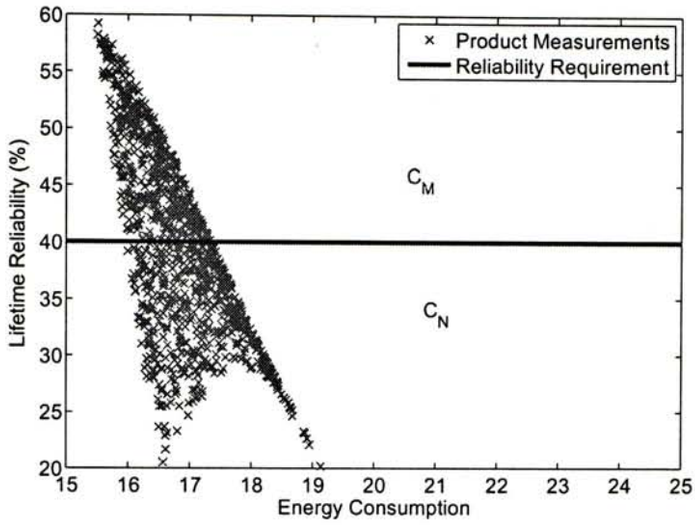
7.5.2 Results and Discussion

We first evaluate the effectiveness of the proposed method, given every task can be allocated onto any processor core. As shown in Table 7.3 and Table 7.4, we compare the solutions obtained by using the proposed design-stage and online strategies. In this table, $\mathbf{E_Y}$[Energy] (Column 2) is the estimation value and available at design stage, indicating the expected value of energy consumption of all products. Column 3-5 are obtained by applying the initial TAS solution on sample products. To be specific, C_N^{init} and C_M^{init} (Column 3 and 4) represent the sets of sample products that cannot meet reliability requirements and those that can, respectively. The expected energy consumption over the products that meet reliability requirement with initial solutions is denoted by $\mathbf{E}_{C_M^{\text{init}}}$[Energy] and acquired in Column 5. Columns 6-9 are achieved by applying the online adjustment on the sample products. In particular, for the products that meet reliability constraints with ini-

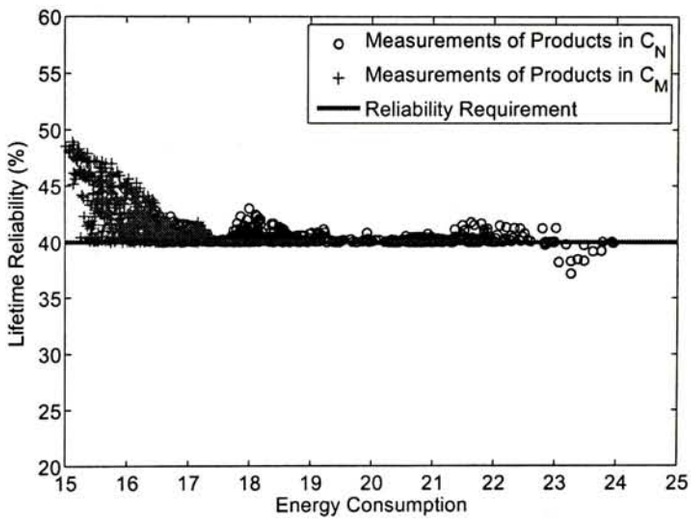


(a) u_{I} (b) u_{II}

Figure 7.6: Description of Usage Strategies.



(a) Initial Solution



(b) Online Adjustment

Figure 7.7: Comparison of Initial Solution and Online Adjustment in Product Measurements.

tial solution, we evaluate their average energy consumption by applying online adjustment and demonstrate the results in Column 8. We also report the expected energy consumption of the products that meet reliability requirements with online adjustment in Column 9.

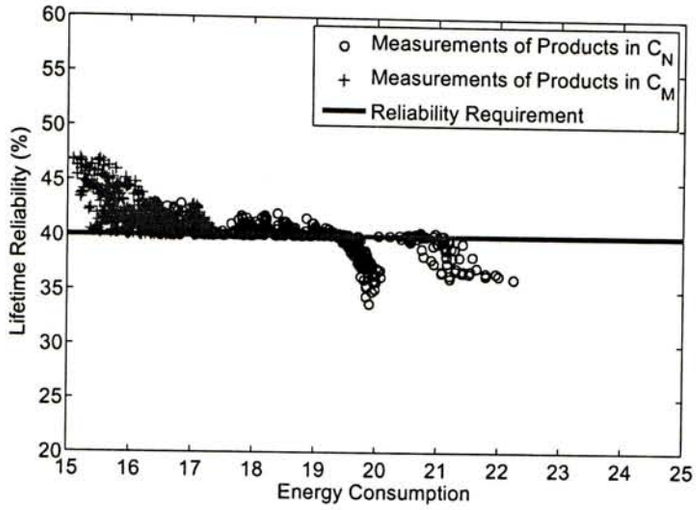
online adjustment					ΔC_M (%)
tasks w. const (%)	C_N^{on} (%)	C_M^{on} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]	$\mathbf{E}_{C_M^{\text{on}}}$[Energy]	
0	1.2	98.8	16.0905	17.6617	53.3
25	16.9	83.1	16.1755	16.9475	37.6
50	25.8	74.2	16.1815	16.6902	28.7
75	**29.2**	**70.8**	**16.3702**	**16.7894**	**25.3**

Table 7.5: Effectiveness of The Proposed Strategy with Mapping Constraints.

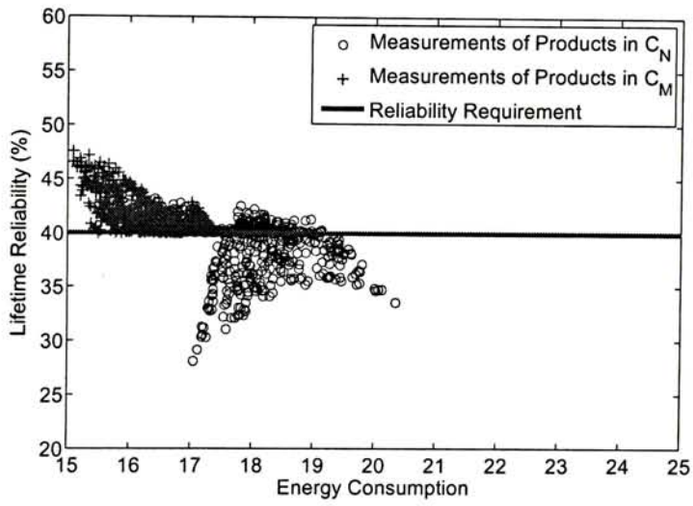
As shown in this table, the online adjustment significantly enhance the lifetime reliability of sample products. The percentage of products that meet reliability requirement with online adjustment is much higher than that with initial solution. For example, in case of $\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_1$, only 45.5% chips are able to meet the reliability requirement that the probability of surviving until the target service life is no less than 40% by applying the initial solution. With online adjustment, by contrast, this value is improved to 98.8% (see Line 3). In average, the proposed online adjustment can lead to 30% more products meeting reliability requirements.

Moreover, the energy consumption of the reliable chips with initial solution (i.e., the chips in set C_N^{init}) can be reduced by using the online adjustment. To take a closer observation, we plot the measurements of all sample products in terms of energy consumption and lifetime reliability on Fig. 7.7, for the case $\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_1$. As demonstrated in this figure, with respect to the chips in set C_M^{init} the proposed online adjustment essentially trades reliability margin for energy reduction. As for the chips in set C_N^{init}, we achieve reliability enhancement by sacrificing some energy. As a result, the expected energy consumption of the products that meet reliability requirements with online adjustment (Column 9) is higher than the design stage estimation (Column 2).

We are also interested in the task allocation and scheduling problems within which some tasks must be allocated on a certain type of processor cores (i.e., task mapping flexibility constraints). In the experiments, we vary the percentage of tasks with such constraints and randomly select a set of tasks. The experimental



(a) Online Adjustment (25% Tasks with Constraints)



(b) Online Adjustment (50% Tasks with Constraints)

Figure 7.8: Comparison of Initial Solution and Online Adjustment in Product Measurements with Mapping Constraints.

results for the case $\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_1$ are depicted in Table 7.5. The characteristics of corresponding initial solution are presented in Row 3 of Table 7.3.

The benefit provided by the proposed online adjustment gradually reduces when more tasks have the task mapping constraints. But still we achieve some reliability enhancement and energy saving. Consider the case that 75% tasks have

mapping constraints (see the last line). The percentage of products that meet reliability requirement increases from 45.5% to 70.8%, and the improvement is as high as 25.3%. This is because, even if most tasks are allocated onto appointed core type, there still exists some flexibility induced by the cores assignment and scheduling order. Based on this observation, we can prepare codes for the tasks that significantly affect the lifetime reliability on various cores only while fix the core assignment of the remaining tasks.

7.6 Conclusion

Because of the usage strategy deviation of modern multi-mode embedded systems, the unified task schedules generated according to the common usage preference may not be reliable or energy-efficient for each individual product. In this paper, we propose a novel customer-aware task allocation and scheduling technique to tackle this problem, wherein initial schedules are generated at design stage and each product is optimized separately with online adjustment at regular intervals. We conduct exhaustive experiments on hypothetical MPSoC platforms to demonstrate the effectiveness of the proposed method.

7.7 Appendix

Proof of 1-1 Correspondence

Clearly, we can always reconstruct a unique task schedule with the given solution representation. We therefore place the primary emphasis on how to construct a unique solution representation with the given task schedule.

A feasible procedure is shown in Fig. 7.9, where we maintain a list of tasks with zero in-degree (denoted by $\mathcal{L}$), and the tasks are sorted in the order of their indexes.

```
1  Initialize L
2  Set the scheduling order sequence as empty
3  Repeat until there is no task in L
4      Repeat for every task in L in index order until a task is selected
5      If the earliest available time of this task on the core indicated
       by resource binding sequence conforms to the schedule
6          Select this task
7          Insert this task at the end of scheduling order sequence
8          Remove this task from task graph
9          Update L
```

Figure 7.9: Construct Representation with the Given Schedule.

Proof of Completeness

We first introduce a few notations before the theoretical proof. If task i must be finished before task j start its execution according to the task graph, the precedence constraint between these two tasks are denoted by $i \prec j$. If there exists no direct or indirect precedence constraint, we have $i \circlearrowright j$. In particular, $i \preceq j$ is used to represent that either $i \prec j$ or $i \circlearrowright j$.

In the following, we theoretically prove that, starting from a valid scheduling order $A = (a_1, a_2, \cdots, a_n)$, we are able to reach any other valid scheduling order $B = (b_1, b_2, \cdots, b_n)$ by finite times of M1.

Fig. 7.10 illustrates a feasible procedure. The operation in Line 5 does not

violate the precedence constraint. This is because, since A is a valid schedule order, we have $a_i \preceq a_{i+1} \preceq \cdots \preceq a_j$. On the other hand, since B is also a valid order, $b_i \preceq b_{i+1} \preceq \cdots \preceq b_n$. Note that, $a_j = b_i$. Thus $a_i \preceq a_{i+1} \preceq \cdots \preceq a_j \preceq b_{i+1} \preceq \cdots \preceq b_n$. In addition, set $\{a_i, a_{i+1}, \cdots, a_{j-1}\} \subseteq$ set $\{b_{i+1}, \cdots, b_n\}$. Consequently, $a_j \circlearrowleft a_{j-1}$, $a_j \circlearrowleft a_{j-2}$, $\cdots$ $a_j \circlearrowleft a_i$.

```
1  Repeat for every task in B from left to right
2      Set current task as b_i
3      Find current task in A, denoted by a_j
4      Repeat until a_j moves to i^th position in A
5          Pick up a_j and apply M1
6          Postprocess sequence A
```

Figure 7.10: Solution Space Exploration.

□ End of chapter.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

With the relentless scaling of semiconductor technology, the reliability issue of multi-core systems has become one of the major concerns for both academics and industry. The research community has observed a new trend that challenges the conventional wisdom: the expected service life of state-of-the-art IC designs becomes much shorter than a few years before. This thesis proposes novel analytical models and techniques to exhaustively explore this problem.

We develop a comprehensive mathematical model to analyze the lifetime reliability of homogeneous multi-core systems with redundant cores. Different from previous work, this model captures all key features of multi-core systems and it is applicable for arbitrary failure distributions. Three special cases and a set of numerical experiments are discussed in detail to show the practical applicability of the proposed approach.

An efficient yet accurate simulation framework is then proposed to facilitate designers to make design decisions that affect system mean time to failure. In this framework, we first simulate the representative workload on processor-based

SoCs in a fine-grained manner and trace the reliability-related factors. The usage strategies are taken as inputs to the simulator. Sequentially, we calculate the single quantity "aging rate" with the traced information. By doing so, we "hide" the impact of reliability-related usage strategies into aging rate. We theoretically prove that the system lifetime reliability can be expressed as a function of aging rate, and that the accuracy is perfectly acceptable. Four case studies are conducted to show the flexibility of effectiveness of the proposed simulation framework.

We also present three novel task allocation and scheduling techniques that explicitly take the lifetime reliability into account. Experimental results on various multiprocessor platforms and task graphs demonstrate the efficacy of the proposed approaches.

8.2 Future Work

Our theory and application effort now focuses on design-stage decisions and only touches on the online adjustment of a specific application (refer to Chapter 7). Even this online adjustment technique, since all feasible solutions are prepared at design stage and stored on-chip beforehand, has essentially limited flexibility. Moreover, the design-stage optimization, albeit well conducted, relies on the prediction of system usage scenario, for example, the application flow characteristics. However, it is extremely hard for the designers to make perfect prediction before launching the products into commercial market. Even if the prediction accuracy is acceptable for a great share of products, it is very likely that each individual product were not optimized with respect to its specific usage strategy. In this sense, it is promising to give processor-based SoCs the ability of self-adjustment.

We can develop an online simulation framework as a breakthrough. To be specific, the simulation framework *AgeSim* introduced in this thesis can be used for characterizing the lifetime reliability of processor-based SoCs with various usage

strategies at the design stage. However, the accuracy of simulation depends on that of the representative workload, which can vary considerably on each particular product. For example, modern electronic products typically combines multiple functions in one. The users might have quite different usage preference and this information is available only if the products are shipped to the customers. Needless to say, because of design complexity it is very difficult, if not impossible, for the designers to optimize each product according to its users' specific usage preference. As a consequence, introducing the self-adjustment capacity onto state-of-the-art processor-based SoCs can be a promising solution.

In addition, various dynamic power/thermal management policies have been proposed in the past decades. Given the lifetime reliability constraint a set of policies are predefined before usage. Since the actual workload might be different from that considered at the design stage, the policies can be adjusted for energy consumption under the same reliability constraint. Similar to the previous example, the adjustment can be performed only when the actual workload becomes available and hence must conducted at the online stage.

Moreover, due to the imperfect manufacturing process, there is significant variation in device parameters (e.g., channel length, threshold voltage) among transistors and hence the performance deviation of processor cores on the same die or across different dies [84]. The occurrence of such random effect on each particular chip cannot be predicted at the design stage but affects the lifetime reliability of the system. For instance, suppose the equal load-sharing scheme stochastically optimizes the system lifetime reliability, that is, the expected mean time to failure of entire product volume is maximized. But if each product is able to adjust its load-sharing scheme according to its specific frequency map, the lifetime reliability can be further improved. Hence, again, we need an automatic reliability management framework for this purpose.

□ **End of chapter.**

Bibliography

[1] Methods for calculating failure rates in units of fits (jesd85). *JEDEC Publication*, 2001.

[2] Failure mechanisms and models for semiconductor devices (jep122c). *JEDEC Publication*, 2003.

[3] T. Adam, K. Chandy, and J. Dickson. A Comparison of List Scheduling for Parallel Processing Systems. *Communications of the ACM*, 17(12):685–690, December 1974.

[4] A. Agarwal and M. Levy. The KILL Rule for Multicore. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 750–753, 2007.

[5] S. V. Amari and R. Bergman. Reliability Analysis of *k*-out-of-*n* Load-Sharing Systems. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 440–445, 2008.

[6] S. V. Amari, K. B. Misra, and H. Pham. Tampered failure rate load-sharing systems: Status and perspectives. In K. B. Misra, editor, *Handbook of Performability Engineering*, pages 291–308. Springer London, 2008.

[7] ARM. ARM11 PrimeXsys Platform. http://www.jp.arm.com/event/images/forum2002/02-print_arm11_primexsys_platform_ian.pdf.

[8] T. M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings International Symposium on Microarchitecture (MICRO)*, pages 196–207, 1999.

[9] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, page 113.2, 2003.

[10] M. D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, 27(6):540–547, June 1978.

[11] S. Bell et al. TILE64 Processor: A 64-Core SoC with Mesh Interconnect. In *Proceedings International Solid State Circuits Conference (ISSCC)*, pages 88–89, 2008.

[12] L. Benini, A. Bogliolo, and G. de Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on VLSI Systems*, 8(3):299–316, June 2000.

[13] L. Benini and G. D. Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Springer-Verlag, 1997.

[14] P. Bjorn-Jorgensen and J. Madsen. Critical Path Driven Cosynthesis for Heterogeneous Target Architectures. In *Proceedings International Conference on Hardware Software Codesign*, pages 15–19, 1997.

[15] J. R. Black. Electromigration - a brief survey and some recent results. *IEEE Transactions on Electron Devices*, ED-16(4):338–347, April 1969.

[16] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, Nov.-Dec. 2005.

[17] S. Borkar. Thousand Core Chips – A Technology Perspective. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 746–749, 2007.

[18] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[19] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *Proceedings International Symposium on High-Performance Computer Architecture (HPCA)*, pages 171–182, 2001.

[20] T. D. Burd and R. W. Brodersen. Design Issues for Dynamic Voltage Scaling. In *Proceedings International Symposium on Low Power Electronics and Design (ISLPED)*, pages 9–14, 2000.

[21] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A Dynamic Voltage Scaled Microprocessor System. *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580, Nov. 2000.

[22] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing*. Kluwer Academic Publishers, 2000.

[23] S.-C. Chang, S.-Y. Deng, and J. Y.-M. Lee. Electrical characteristics and reliability properties of metal-oxide-semiconductor field-effect transistors with dy2o3 gate dielectric. *Applied Physics Letters*, 89(5), August 2006.

[24] I.-R. Chen and F. B. Bastani. Warm standby in hierarchically structured process-control programs. *IEEE Transactions on Sofeware Engineering*, 20(8):1994, August 1994.

[25] Cisco. Cisco and IBM Collaborate to Design and Build World's Most Sophisticated, High-Performance 40Gbps Custom Chip. http://newsroom.cisco.com/dlls/partners/news/2004/pr_prod_06-09.html.

[26] J. Clabes et al. Design and Implementation of the POWER5 Microprocessor. In *Proceedings International Solid State Circuits Conference (ISSCC)*, pages 56–57, 2004.

[27] J. Cong and K. Gururaj. Energy Efficient Multiprocessor Task Scheduling under Input-dependent Variation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 411–416, 2009.

[28] R. C. Correa, A. Ferreira, and P. Rebreyend. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):825–837, August 1999.

[29] A. Coskun, T. Rosing, K. Mihic, G. D. Micheli, and Y. L. Lebici. Analysis and optimization of mpsoc reliability. *Journal of Low Power Electronics*, 15(2):159–172, February 2006.

[30] A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature Aware Task Scheduling in MPSoCs. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1659–1664, 2007.

[31] A. Dasgupta and R. Karri. Electromigration Reliability Enhancement Via Bus Activity Distribution. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 353–356, 1996.

[32] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task Graphs for Free. In *Proceedings International Conference on Hardware Software Codesign*, pages 97–101, 1998.

[33] A. Dogan and F. Ozguner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):308–323, March 2002.

[34] C. E. Ebeling. In *An Introduction to Reliability and Maintainability Engineering*. Waveland Press, 2005.

[35] X. Fu, T. Li, and J. Fortes. NBTI Tolerant Microarchitecture Design in the Presence of Process Variation. In *Proceedings International Symposium on Microarchitecture (MICRO)*, pages 399–410, 2008.

[36] D. Geer. Chip Makers Turn to Multicore Processors. *IEEE Computer*, 38(5):11–13, May 2005.

[37] J. E. Gentle. In *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, 2nd edition, 2003.

[38] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):686–701, June 1993.

[39] A. K. Goel. In *High-speed VLSI Interconnections*. IEEE Press, 2nd edition, 2007.

[40] L. R. Goel, R. Gupta, and R. K. Agnihotri. Analysis of a three-unit redundant system with two types of repair and inspection. *Microelectron Reliability*, 29(5):769–773, 1989.

[41] Z. Gu, C. Zhu, L. Shang, and R. P. Dick. Application-specific mpsoc reliability optimization. *IEEE Transactions on VLSI Systems*, 16(5):603–608, May 2008.

[42] T. F. Hassett, D. L. Dietrich, and F. Szidarovszky. Time-varying failure rates in the availability and reliability analysis of repairable systems. *IEEE Transactions on Reliability*, 44:155–160, March 1995.

[43] S. Herbert and D. Marculescu. Characterizing Chip-Multiprocessor Variability-Tolerance. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 313–318, 2008.

[44] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *IEEE Computer*, 41(7):33–38, July 2008.

[45] C.-K. Hu, R. Rosenberg, H. S. Rathore, D. B. Nguyen, and B. Agarwala. Scaling Effect on Electromigration in On-Chip Cu Wiring. In *Proceedings IEEE Interconnect Technology International Conference*, pages 267–269, 1999.

[46] L. Huang and Q. Xu. On Modeling the Lifetime Reliability of Homogeneous Manycore Systems. In *Proceedings Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 87–94, 2008.

[47] L. Huang and Q. Xu. AgeSim: A Simulation Framework for Evaluating the Lifetime Reliability of Processor-Based SoCs. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 51–56, 2010.

[48] L. Huang and Q. Xu. Characterizing the Lifetime Reliability of Manycore Processors with Core-Level Redundancy. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 680–685, 2010.

[49] L. Huang and Q. Xu. Energy-Efficient Task Allocation and Scheduling for Multi-Mode MPSoCs under Lifetime Reliability Constraint. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1584–1589, 2010.

[50] L. Huang and Q. Xu. Lifetime reliability for load-sharing redundant systems with arbitrary failure distributions. *IEEE Transactions on Reliability*, 59(2):319–330, June 2010.

[51] L. Huang, R. Ye, and Q. Xu. Customer-Aware Task Allocation and Scheduling for Multi-Mode MPSoCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, 2011.

[52] L. Huang, F. Yuan, and Q. Xu. Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 51–56, 2009.

[53] L. Huang, F. Yuan, and Q. Xu. On task allocation and scheduling for lifetime extension of platform-based mpsoc designs. *IEEE Transactions on Parallel and Distributed Systems*, 2011.

[54] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Thermal-Aware Task Allocation and Scheduling for Embedded Systems. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 898–899, 2005.

[55] IBM. Cell Broadband Engine Programming Handbook Including PowerXCell 8i. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs /1741C509C5F64B3300257460006FD68D/$file /CellBE_PXCell_Handbook_v1.11_12May08_pub.pdf.

[56] IBM. IBM PowerPC 750CL Microprocessor Revision Level DD2.x. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs /2F33B5691BBB8769872571D10065F7D5/$file /750cldd2x_ds_v2.4_pub_29May2007.pdf.

[57] Intel. SA-1100 Microprocessor Technical Reference Manual, 1998.

[58] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proceedings International Symposium on Low Power Electronics and Design (ISLPED)*, pages 197–202, 1998.

[59] R. Jejurikar, C. Pereira, and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 275–280, 2004.

[60] A. Jerraya, H. Tenhunen, and W. Wolf. Multiprocessor Systems-on-Chips. *IEEE Computer*, 38(7):36–40, July 2005.

[61] N. K. Jha. Low Power System Scheduling and Synthesis. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 259–263, 2001.

[62] S. Kajihara, K. Ishida, and K. Miyase. Reliability Modeling and Management in Dynamic Microprocessor-Based Systems. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 1057–1060, 2006.

[63] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Multi-Mechanism Reliability Modeling and Management in Dynamic Systems. *IEEE Transactions on VLSI Systems*, 16(4):476–487, April 2008.

[64] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Multi-Mechanism Reliability Modeling and Management in Dynamic Systems. *IEEE Transactions on VLSI Systems*, 16(4):476–487, April 2008.

[65] V. Kianzad and S. S. Bhattacharyya. CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems. In *Proceedings IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 28–40, 2004.

[66] I. Koren and C. M. Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann, 2007.

[67] W. Kuo and M. J. Zuo. In *Optimal Reliability Modeling: Principles and Applications*. John Wiley & Sons, 2003.

[68] Y.-K. Kwok and I. Ahmad. Static task scheduling and allocation algorithms for scalable parallel and distributed systems: Classification and performance comparison. In Y. C. Kwong, editor, *Annual Review of Scalable Computing*, pages 107–227. 2000.

[69] G. Liao, E. R. Altman, V. K. Agarwal, and G. R. Gao. A Comparative Study of Multiprocessor List Scheduling Heuristics. In *Proceedings Hawaii International Conference on System Sciences*, pages 68–77, 1994.

[70] H.-H. Lin, K.-H. Chen, and R.-T. Wang. A multivariate exponential shared-load model. *IEEE Transactions on Reliability*, 42(1):165–171, March 1993.

[71] M. Lin and L. T. Yang. Genetics algorithms for scheduling real-time tasks onto multi-processors. In Y.-S. Dai, Y. Pan, and R. Raje, editors, *Distributed Computing: Evaluation, Improvement and Practice*, pages 213–238. Nova Science Publishers, 2007.

[72] H. Liu. Reliability of a load-sharing k-out-of-n:g system: Non-iid components with arbitrary distributions. *IEEE Transactions on Reliability*, 47(3):279–184, September 1998.

[73] Z. Lu, W. Huang, M. R. Stan, K. Skadron, and J. Lach. Interconnect lifetime prediction for reliability-aware systems. *IEEE Transactions on VLSI Systems*, 15(2):159–172, February 2007.

[74] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 721–725, 2002.

[75] F. P. Mathur and A. Avizienis. Reliability Analysis and Architecture of a Hybrid Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair. In *Proceedings AFIPS Conference, Spring Joint Computer Conference*, pages 375–383, 1970.

[76] M. Nicolaidis. Design for soft error mitigation. *IEEE Transactions on Device and Materials Reliability*, 5(3):405–418, September 2005.

[77] Nvidia. Geforce 8800 graphics processors. http://www.nvidia.com/page /geforce 8800.html.

[78] Nvidia Provides Second Quarter Fiscal 2009 Business Update. http://www.nvidia.com/object/io_1215037160521.html.

[79] H. Oh and S. Ha. Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-Time Constraints. In *Proceedings IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 133–138, 2002.

[80] J. Oh and C. Wu. Genetic-algorithm-based real-time task scheduling with multiple goals. *Journal of Systems and Software*, 71(3):245–258, May 2004.

[81] Philips. Nexperia Processor. http://www.semiconductors.philips.com /products/nexperia.

[82] T. S. Rosing, K. Mihic, and G. D. Micheli. Power and Reliability Management of SoCs. *IEEE Transactions on VLSI Systems*, 15(4):391–403, April 2007.

[83] A. Sangiovanni-Vincentelli, L. Carloni, F. D. Bernardinis, and M. Sgroi. Benefits and Challenges for Platform-based Design. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 409–414, 2004.

[84] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, February 2008.

[85] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Cosynthesis of Energy-Efficient Multimode Embedded Systems With Consideration of Mode-Execution Probabilities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):153–169, February 2005.

[86] J. Shao and L. R. Lamberson. Modeling a shared-load k-out-of-n: G system. *IEEE Transactions on Reliability*, 40(2):205–209, June 1991.

[87] S. M. Shatz, J.-P. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Transactions on Computers*, 41(9):1156–1168, September 1992.

[88] J. She and M. G. Pecht. Reliability of a k-out-of-n warm-standby system. *IEEE Transactions on Reliability*, 41(1):72–75, March 1992.

[89] D. J. Sherwin and A. Bossche. *The Reliability, Availability and Productiveness of Systems*. Chapman & Hall, 1993.

[90] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A Proactive Wearout Recovery Approach for Exploiting Microarchitectural Redundancy to Extend Cache SRAM Lifetime. In *Proceedings IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pages 353–362, 2008.

[91] J. Shin, V. Zyuban, Z. Hu, J. Rivers, and P. Bose. A Framework for Architecture-Level Lifetime Reliability Modeling. In *Proceedings IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 534–53, 2007.

[92] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pages 2–13, 2003.

[93] Sony Computer Entertainment Inc. Cell Broadband Engine. http://cell.scei.co.jp/.

[94] SquareTrade. Report on Xbox 360 Failure Rates. http://blog.squaretrade.com/2008/02/xbox-fail-rates.html.

[95] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *Proceedings IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pages 276–287, 2004.

[96] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Impact of Technology Scaling on Lifetime Reliability. In *Proceedings IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2004.

[97] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting Structural Duplications for Lifetime Reliability Enhancement. In *Proceedings*

IEEE/ACM International Symposium on Computer Architecture (ISCA), pages 520–531, 2005.

[98] S. Srinivasan and N. K. Jha. Safety and reliability driven task allocation in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(3):238–251, March 1999.

[99] J. H. Stathis. Reliability limits for the gate insulator in cmos technology. *IBM Journal of Research and Development*, 46(2/3):265–283, 2002.

[100] J. Staunstrup and W. Wolf, editors. *Hardware/Software Co-Design: Principles and Practice*. Kluwer Academic Publishers, 1997.

[101] K. Stavrou and P. Trancoso. Thermal-Aware Scheduling: A Solution for Future Chip Multiprocessors Thermal Problems. In *Proceedings EUROMICRO Conference on Digital System Design (DSD)*, pages 123–126, 2006.

[102] R. Subramanian and V. Anantharaman. Reliability analysis of a complex standby redundant system. *Reliability Engineering and System Safety*, 48:57–70, 1995.

[103] Tilera. Tile64 processor family. http://www.tilera.com/products/processors.php.

[104] A. Tiwari and J. Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *Proceedings International Symposium on Microarchitecture (MICRO)*, pages 129–140, 2008.

[105] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W.-L. Hung. Reliability-Centric Hardware/Software Co-design. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 375–380, 2005.

[106] K. S. Trivedi. In *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, second edition, 2002.

[107] E. J. Vanderperre. Reliability analysis of a warm standby system with general distributions. *Microelectron Reliability*, 30(3):487–490, 1990.

[108] S. Vangal et al. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In *Proceedings International Solid State Circuits Conference (ISSCC)*, pages 98–99, 2007.

[109] B. Vermeulen, S. Oostdijk, and F. Bouwman. Test and Debug Strategy of the PNX8525 Nexperia^{TM} Digital Video Platform System Chip. In *Proceedings IEEE International Test Conference (ITC)*, pages 121–130, Baltimore, MD, Oct. 2001.

[110] M. Xie, Y.-S. Dai, and K.-L. Poh. In *Computing Systems Reliability: Models and Analysis*. Kluwer Academic Publishers, 2004.

[111] Y. Xie and W.-L. Hung. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design. *Journal of VLSI Signal Processing*, 45:177–189, 2006.

[112] Q. Xu and N. Nicolici. Resource-Constrained System-on-a-Chip Test: A Survey. *IEE Proceedings, Computers and Digital Techniques*, 152(1):67–81, January 2005.

[113] B. S. Yoo and C. R. Das. A fast and efficient processor allocation scheme for mesh-connected multicomputers. *IEEE Transactions on Computers*, 51(1):46–60, January 2002.

[114] S. Zafar, A. Kumar, E. Gusev, and E. Cartier. Threshold Voltage Instabilities in High-κ Gate Dielectric Stacks. *IEEE Transactions on Device and Materials Reliability*, 5(1):45–64, March 2005.

[115] S. Zafar, A. Kumar, E. Gusev, and E. Cartier. Threshold voltage instabilities in high-κ gate dielectric stacks. *IEEE Transactions on Device and Materials Reliability*, 5(1):45–64, March 2005.

[116] S. Zhang and K. S. Chatha. Approximation Algorithm for the Temperature-Aware Scheduling Problem. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 281–288, 2007.

[117] T. Zhang, M. Xie, and M. Horigome. Availability and reliability of k-out-of-$(m+n)$:g warm standby systems. *Reliability Engineering and System Safety*, 91:381–387, 2006.

[118] C. Zhu, Z. Gu, R. P. Dick, and L. Shang. Reliable Multiprocessor System-on-Chip Synthesis. In *Proceedings IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, pages 239–244, 2007.